НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6. 050201 «Системна інженерія»

на тему: «Система автоматизованого тестування веб-програмного забезпечення»

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-з51
Плющ Леонід Вікторович _____

Керівник:
Зав. кафедри, д.т.н, професор Ролік О. І.
_____

Рецензент:

_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6. 050201 «Системна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

ЗАВДАННЯ
на дипломний проект студенту
Плющу Леоніду Вікторовичу

1.Тема проекту «Система автоматизованого тестування веб-програмного забезпечення», керівник проекту Ролік Олександр Іванович, д.т.н., професор, затверджені наказом по університету від «___»_____ 2019 р. №_____

2. Термін подання студентом проекту _____

3. Вихідні дані до проекту
_____
_____

4. Зміст пояснювальної записки
_____
_____

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)
_____
_____-
_____

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
	Огляд існуючих рішень	27.04.2019	
	Вибір алгоритму та інструментарію	11.05.2019	
	Розробка системи тестування	24.05.2019	
	Оформлення пояснювальної записки	01.06.2019	

Студент Л. В. Плющ

Керівник проекту О. І. Ролік

АНОТАЦІЯ

Плющ Л. В. Система автоматизованного тестування веб – програмного забезпечення. КПІ імені Ігоря Сікорського, Київ, 2019.

Проект містить 67 с. тексту, 28 рисунків, 4 таблиці, посилання на 33 літературні джерела.

Ключові слова: автоматизована система, програмне забезпечення, веб-додаток, тестування.

Об’єктом розробки є система автоматизованного тестування веб-додатку.

Мета розробки – запропонування алгоритму та засобу для підвищення ефективності тестування веб-додатків.

У дипломному проекті розроблено автоматизовану систему тестування Web-додатків яка використовує метод попарного тестування. Обрана техніка дозволяє позбавитися надлишкових перевірок та виявляти найбільшу кількість помилок при найменшої кількості тестів.

Також розроблено приклад тестів та здійснено перевірку працездатності та продуктивності системи.

SUMMARY

Pliushch L. V. System of automated testing of web software. Igor Sikorsky KPI, Kyiv, 2019.

The project contains 67 pages of text, 28 figures, 4 tables, links to 33 literary sources.

Keywords: automated system, software, web application, testing.

The object of development is automated system of web-application testing.

The purpose of development is to provide algorithm and utility for effective web-application testing.

Within graduation project an automated testing system for Web applications was developed. This system implements Pairwise Testing method. The chosen technique allows to get rid of excess checks and to find the greatest number of errors with the smallest number of tests.

An example of test has been developed and performance testing has been performed.

Пояснювальна записка
до дипломного проекту
на тему: «Система автоматизованого тестування веб-програмного забезпечення»

Київ – 2019 рік

№рядка	Формат	Познака	Найменування	Аркушів	№ екз.	Примітка
1			Документація загальна			
2						
3			Розроблена заново			
4						
5	А4	ІА51.010БАК.000 ОП	Завдання на дипломне	1		
6			проектування			
7			Анотація українською мовою	1		
8			Анотація іноземною мовою	1		
9			Відомість технічного проекту	1		
10						
11	А4	ІА51.010БАК.000 ПЗ	Пояснювальна записка	67		
12						
13						
14						
15						
16						
17						
18						
22						
23						
24						
25						
26						
27						

Зм.	Арк.	№ док.	Підпис	Дата				
Розроб.		Плющ Л.В.				Літ.	Аркуш	Аркушів
Перев.						Т	1	1
Тех.конт.						Група ІА-з51		

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПЗ	програмне забезпечення
ПК	персональний комп’ютер
DDA	disability discrimination act
DSN	data source name
ODBC	open database connectivity
ODT	object driven testing
RAD	rapid application development

ВСТУП

Актуальність дослідження. Нові методології програмування, такі як прискорена розробка додатків (RAD), екстремальне програмування, призвели до інтенсивних розробок засобів автоматизованого тестування. Головною особливістю цих методологій є можливість отримання різних версій програмного продукту все зростаючого обсягу з високою частотою. Звідси, сучасне тестування набуло ітеративну природу – кожна нова версія супроводжується значною кількістю нових тестів, а так само переробкою існуючих автоматизованих засобів тестування. Оскільки для кожної версії проекту необхідно розробити засіб дозволяючий найбільш повно протестувати всі його особливості, актуальні методи і алгоритми проектування автоматизованих засобів тестування за допомогою яких можна швидко і з найменшими витратами розробити подібний засіб. Відмінні особливості проектування автоматизованих засобів тестування випливають зі специфіки завдань які ставляться перед ними:

- Об'єктивна трудність тестування: це деструктивний, тобто зворотний творчому, процес. Тому, проектовані засоби повинні замість збору та обробки інформації виконувати її розбиття на частини і проводити аналіз цих частин.
- Тестування абсолютно всіх можливих ситуацій нескінченно велике, тому проектовані засоби тестування повинні враховувати неможливість перебору всіх можливих тестів.

У свою чергу, при розробці програмних продуктів, з метою зменшення витрат широко застосовуються методи і алгоритми проектування програмних або інформаційних систем. Але не всі вони підходять для проектування засобів тестування. Крім того, в умовах швидкої і постійної зміни тестованого проекту доводиться постійно перепроектовувати існуючі засоби.

Інтенсивна розробка та модернізація автоматизованих засобів тестування робить актуальним завдання пошуку та застосування найбільш оптимальних методів і алгоритмів проектування, за допомогою яких можна швидко і з мінімальними витратами спроектувати новий засіб тестування.

Мета та завдання. Метою даної дипломної роботи виступає дослідження методів та алгоритмів автоматизації тестування Web-додатків, а також розробка системи автоматизованого тестування Web-додатків.

За для досягнення поставленої мети у роботі необхідно виконати низку завдань:

- розкрити теорію тестування Web-додатків;
- провести огляд засобів і методів автоматизації тестування Web-додатків;
- запропонувати цільовий аналіз системи автоматизованого тестування Web-додатків;
- визначити алгоритми системи автоматизованого тестування Web-додатків;
- здійснити вибір інструментарію автоматизованого тестування Web-додатків;
- розробити систему автоматизованого тестування Web-додатків;
- провести тестування системи автоматизованого тестування Web-додатків;
- викласти отримані результати дослідження.

Об'єкт та предмет роботи. Об'єктом роботи виступають програмні комплекси щодо тестування веб-додатків. Предметом роботи є автоматизація тестування Web-додатків.

Методи дослідження. Для вирішення поставлених завдань використовувалися методи наукового пізнання, такі як абстракція, порівняння, узагальнення, аналогія, індукція, дедукція.

Практична значущість результатів. У роботі досліджено методи та засоби автоматизованого тестування Web-додатків для користувальницької експлуатації. Отримані результати можуть бути використані у межах автоматизованого тестування Web-додатків.

Структура дипломної роботи. Робота складається зі вступу, 3 розділів, що включають в себе дев’ять підрозділів, висновків, списку літератури з 33 найменувань. Загальний обсяг роботи 67 сторінки.

РОЗДІЛ 1
ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ ТЕСТУВАННЯ WEB-ДОДАТКІВ

1.1 Web-додаток: сутність поняття

Web-додаток являє собою Web-сайт, на якому розміщені сторінки з частково або повністю несформованим вмістом. Остаточний вміст формується тільки після того, як відвідувач сайту запросить сторінку з Web-сервера. У зв'язку з тим що остаточний вміст сторінки залежить від запиту, створеного на основі дій користувача, така сторінка називається динамічною.

Використання Web-додатків приносить певну користь як відвідувачам Web-сайтів, так і їх розробникам.

Web-додатки дозволяють відвідувачам швидко і легко знаходити необхідну інформацію на веб-сайтах з великим об'ємом інформації.

Даний вид Web-додатків дозволяє здійснювати пошук у вмісті, упорядковувати вміст і переміщатися по ньому зручним для відвідувачів способом. Прикладами таких додатків можуть служити внутрішні мережі компаній - Microsoft MSDN (www.msdn.microsoft.com) і Amazon (www.amazon.com). [1]

Web-додатки дозволяють збирати, зберігати і аналізувати дані, отримані від відвідувачів сайту.

Довгий час використовувався метод, при якому дані, введені в HTML-форми, відсилалися для обробки CGI-додатків або спеціально призначеним працівникам у вигляді повідомлень електронної пошти. Web-додаток дозволяє зберігати дані безпосередньо в базі даних, а також отримувати дані і формувати звіти на основі отриманих даних для аналізу. Як приклад можна привести інтерактивні сторінки банків, сторінки для контролю товарних запасів, соціологічні дослідження та опитування, а також форми для

зворотного зв'язку з користувачами.

Web-додаток може використовуватися для оновлення Web-сайтів з періодично мінливим вмістом.

Web-додаток звільняє веб-дизайнера від рутинної роботи постійного оновлення HTML-сторінок сайту. Постачальники вмісту, наприклад редактори новин, відповідають за наявність свіжого матеріалу, а Web-додаток стежить за автоматичним оновленням сайту. Як приклад можна привести Web-версію журналу «The Economist» (www.economist.com) і служби новин CNN (www.cnn.com).

Будь-який Web-додаток являє собою набір статичних і динамічних сторінок. Статична Web-сторінка – це сторінка, яка завжди відображається перед користувачем в незмінному вигляді. Сервер відправляє сторінку за запитом Web-браузера без будь-яких змін. На противагу цьому, сервер вносить зміни в динамічну сторінку перед відправкою її браузеру. У зв'язку з тим що сторінка змінюється, вона називається динамічною. [2]

Статичний Web-сайт містить набір відповідних HTML-сторінок і файлів, розміщених на комп'ютері, на якому встановлено Web-сервер.

Web-сервер – це програмне забезпечення, яке надає Web-сторінки у відповідь на запити клієнта. Зазвичай запит сторінки створюється при натисканні посилання на веб-сторінці, виборі закладки в браузері або введенні URL-адреси сторінки в адресному рядку браузера.

Остаточний вміст статичної сторінки визначається розробником і залишається незмінним у процесі запиту. Приклад вмісту статичної Web-сторінки:

```
<html>
    <head>
        <title>Trio Motors Information Page</title>
    </head>
    <body>
        <h1>About Trio Motors</h1>
        <p>Trio Motors is a leading automobile manufacturer.</p>
    </body>
</html>
```

Весь HTML-код створюється розробником до того моменту, коли сторінка буде розміщена на сервері. Оскільки HTML-код не змінюється після розміщення сторінки на сервері, дана сторінка називається статичною.

Коли Web-сервер отримує запит на видачу статичної сторінки, то, після аналізу запиту, він знаходить потрібну сторінку і відправляє її браузеру, як показано на рис. 1.1.

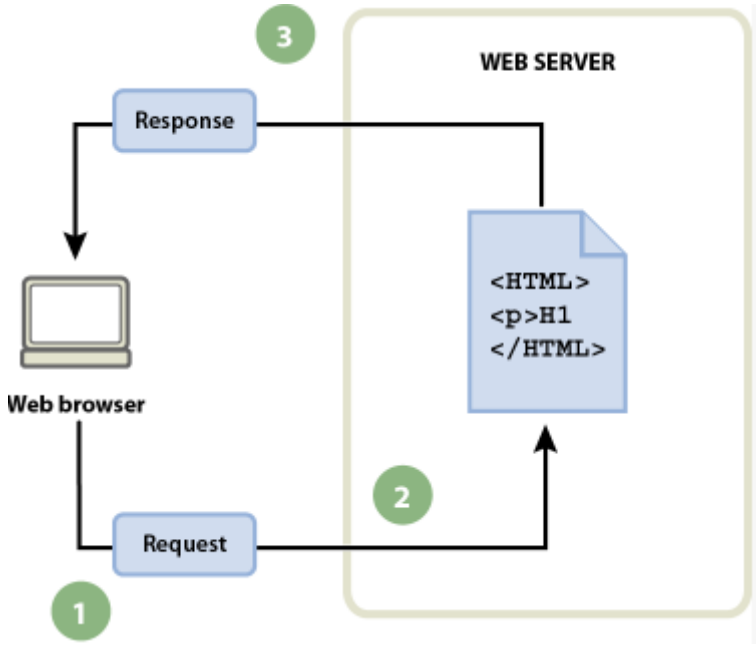


Рисунок 1.1 – Обробка статичної веб-сторінки.

1. Web-браузер запитує статичну сторінку. 2. Web-сервер знаходить сторінку. 3. Web-сервер відправляє сторінку запити її браузеру.

У разі Web-додатків деякі ділянки коду сторінки відсутні до моменту запиту сторінки відвідувачем. Відсутній код формується за допомогою деякого механізму і тільки після цього сторінка може бути відправлена браузеру. [2]

Коли Web-сервер отримує запит на видачу статичної сторінки, він відправляє її безпосередньо браузеру. Однак, коли запитується динамічна сторінка, дії сервера не настільки однозначні. Сервер передає сторінку

спеціальною програмою, яка і формує остаточну сторінку. Така програма називається сервером додатків.

Сервер додатків виконує читання коду, що знаходиться на сторінці, формує остаточну сторінку відповідно до прочитаного коду, а потім видаляє його зі сторінки. В результаті всіх цих операцій виходить статична сторінка, яка передається Web-серверу, який в свою чергу відправляє її клієнтському браузеру. Всі сторінки, які отримує браузер, містять тільки HTML-код. Схематичне зображення процесу наведено на рис. 1.2.

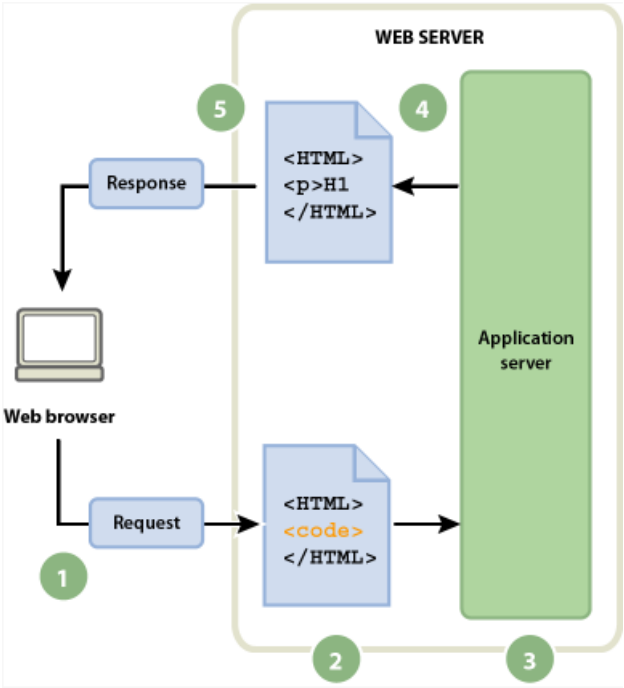


Рисунок 1.2 – Обробка динамічних сторінок.

Обробка динамічних сторінок здійснюється у такому порядку:

1) Web-браузер запитує динамічну сторінку.

2) Web-сервер знаходить сторінку і передає її серверу додатків.

3) Сервер додатків переглядає сторінку на наявність інструкцій і виконує її створення.

4) Сервер додатків повертає підготовлену сторінку на Web-сервер.

5) Web-сервер відправляє підготовлену сторінку запити її браузеру.

Сервер додатків надає можливість використовувати такі ресурси сервера, як бази даних. Наприклад, динамічна сторінка може містити програмні інструкції для сервера додатків, слідуючи яким серверу необхідно отримати певні дані з бази даних і помістити їх в HTML-код сторінки.

Зберігання вмісту в базі даних дозволяє відокремити оформлення веб-сайту від вмісту, яке будуть бачити користувачі. Замість того щоб створювати всі сторінки у вигляді окремих HTML-файлів, пишуться тільки шаблони сторінок для кожного виду інформації, що представляється. Потім вміст завантажується в базу даних, після чого сайт буде витягувати його при запитах користувачів. Крім того, можна оновити інформацію в одному джерелі і продублювати цю зміну на всіх сайтах без редагування кожної сторінки вручну. Adobe Dreamweaver дозволяє створювати Web-форми для вставки, оновлення та видалення інформації в базі даних.

Програмна інструкція, призначена для отримання даних з бази даних, називається запитом до бази даних. Запит складається з критеріїв пошуку, виражених за допомогою мови баз даних, званої SQL (мова структурованих запитів). Текст SQL-запиту розташовується в сценаріях сторінок на стороні сервера або в тегах. [4]

Сервер додатків не може безпосередньо отримати дані з бази, оскільки бази даних використовують специфічні формати зберігання даних. Тому для підключення до бази даних сервер додатків використовує драйвер бази даних - програмний модуль, за допомогою якого встановлюється взаємодія між сервером додатків і базою даних.

Після того як драйвер встановить з'єднання, виконується запит до бази, в результаті чого формується набір записів. Набір записів є безліч даних, отриманих з однієї або декількох таблиць бази даних. Набір записів повертається сервера додатків, який використовує отримані дані для формування сторінки.

Приклад простого запиту до бази даних на мові SQL:

```
SELECT lastname, firstname, fitpoints
FROM employees
```

За допомогою цієї інструкції формується набір записів з трьох стовпців, що містять прізвище (lastname), ім'я (firstname) та набрані бали (fitpoints) всіх співробітників (employees), відомості про яких зберігаються в базі даних.

Рис. 1.3 демонструє процес виконання запиту до бази даних і повернення отриманих даних браузеру.

Для використання у Web-додатку придатна будь-яка база даних за умови, що на сервері встановлений відповідний драйвер.

Для створення малобюджетних додатків можна використовувати файлову базу даних, наприклад базу даних створену за допомогою Microsoft Access. Якщо планується створення надійних корпоративних додатків, рекомендується використовувати серверну базу даних, наприклад, на основі серверів Microsoft SQL Server, Oracle 9i або MySQL. [4]

Якщо база даних і Web-сервер розташовуються на різних комп'ютерах, слід забезпечити швидкісне підключення між системами, оскільки від цього буде залежати ефективність і швидкість роботи всього Web-додатку.

Процес розробки динамічних сторінок складається з написання базового HTML-коду і подальшого створення серверних сценаріїв або тегів HTML-сторінки, за допомогою яких сторінка стає динамічною. Якщо поглянути на остаточний код, видно, що мова сценаріїв вбудована в HTML-код сторінки. Відповідно, такі мови сценаріїв називають мовами, вбудованими в HTML.

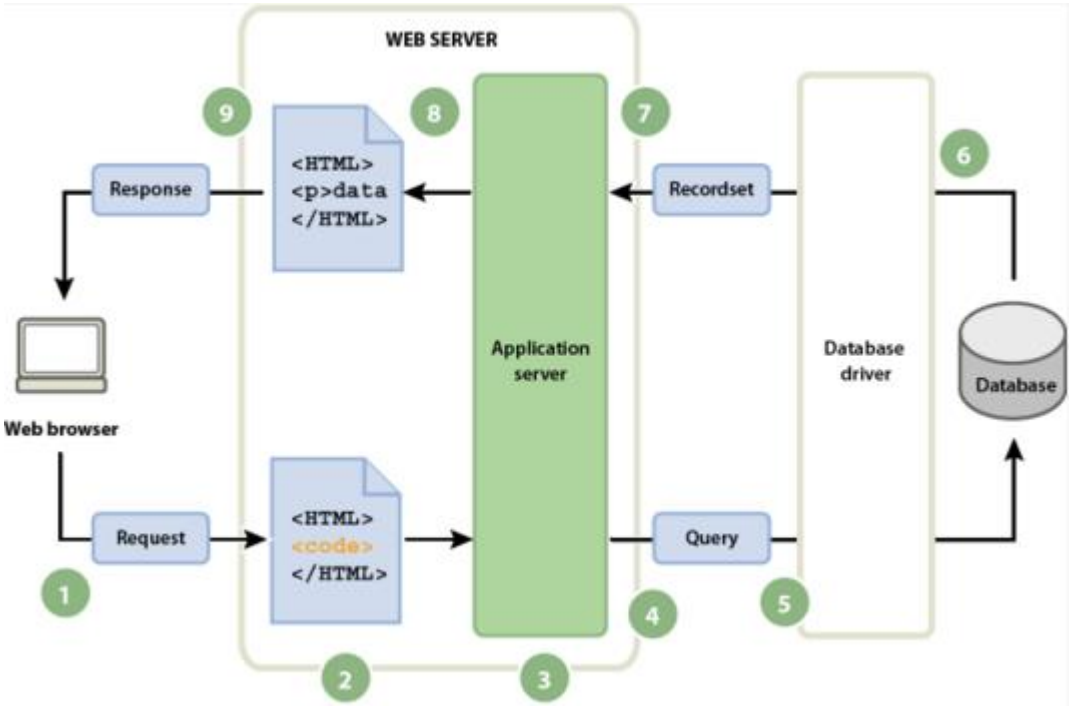


Рисунок 1.3 – Процес виконання запиту до бази даних і повернення отриманих даних браузеру.

1) Web-браузер запитує динамічну сторінку.
2) Web-сервер знаходить сторінку і передає її серверу додатків.
3) Сервер додатків переглядає сторінку на наявність інструкцій і виконує її підготовку.
4) Сервер додатків відправляє запит драйверу бази даних.
5) Драйвер виконує запит в базі даних.
6) Драйвер повертає набір записів.
7) Драйвер передає набір записів серверу додатків.
8) Сервер додатків вставляє дані в сторінку і передає сторінку Web-серверу.
9) Web-сервер відправляє підготовлену сторінку запиту її браузеру.

У наступному прикладі використовується мова розмітки ColdFusion Markup Language (CFML):

```
<html>
    <head>
        <title>Trio Motors Information Page</title>
    </head>
    <body>
        <h1>About Trio Motors</h1>
        <p>Trio Motors is a leading automobile manufacturer.</p>
        <!--- embedded instructions start here --->
        <cfset department="Sales">
        <cfoutput>
        <p>Be sure to visit our #department# page.</p>
        </cfoutput>
        <!--- embedded instructions end here --->
    </body>
</html>
```

Вбудовані в дану сторінку інструкції виконують такі дії:

- створюється змінна з ім'ям «department», після чого їй присвоюється значення рядка «Sales»;
- значення «Sales» поміщається в HTML-код;
- сервер додатків повертає наступну сторінку на Web-сервер:

```
<html>
    <head>
        <title>Trio Motors Information Page</title>
    </head>
    <body>
        <h1>About Trio Motors</h1>
        <p>Trio Motors is a leading automobile manufacturer.</p>
        <p>Be sure to visit our Sales page.</p>
    </body>
</html>
```

Таким чином, згідно проведеного дослідження варто зазначити, що Web-додаток це веб-сайт, на якому розміщені сторінки з частково або повністю невизначеним вмістом. Остаточний вміст таких сторінок формується тільки після того, як відвідувач сайту запросить сторінку з

Web-сервера. У зв'язку з тим що остаточний вміст сторінки залежить від запиту, створеного на основі дій користувача, така сторінка називається динамічною.

1.2 Особливості тестування Web-додатків

Web-додатки – це сфера, що динамічно розвивається. Не всі підходи і методи, що застосовуються для тестування класичних додатків можуть бути застосовні для тестування web-додатків.

Web-додаток – клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером Web-сервер, що вже є по суті двома окремими програмами, які необхідно тестувати як окремо, так і в зв'язці.

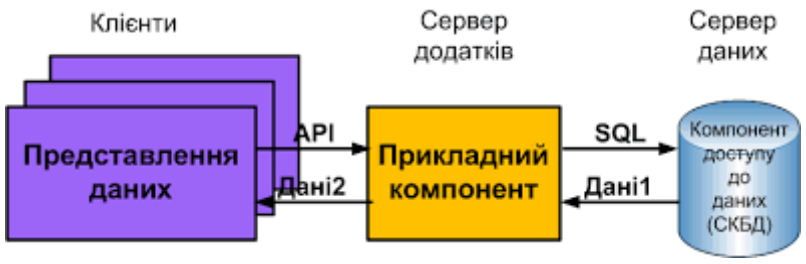


Рисунок 1.4 – Структура web-додатку [6].

Майже всі сучасні програми орієнтовані на роботу з мережею. Зберігання даних Web-додатків здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі. Коли користувач бачe помилку в мережевому середовищі, то часто складно точно вказати, де саме вона відбулася, і тому режим роботи, або повідомлення про помилку, яке ми отримуємо, може бути результатом помилок, що сталися в різних частинах мережевої системи.

Маючи багато спільного з тестуванням класичних програм, тестування Web-орієнтованих додатків має свої особливості, пов'язані насамперед із середовищем функціонування. Маючи компонентні, структурні та

технологічні особливості, Web-додатками притаманні особливості режимів роботи, інсталяції, запуску, зупинки і видалення, а також формування інтерфейсів. Працюючи завжди з мережею і з великою кількістю користувачів, Web-додатки мають на увазі під собою різні права доступу для різних користувачів.

Логіка Web-додатку розподілена між сервером і клієнтом, зберігання даних здійснюється на сервері, обмін інформацією відбувається по мережі.

Одним з переваг підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому Web-додатки є міжплатформними сервісами [7].

Особливості тестування Web-додатків

Технологічні відмінності. Класичний додаток працює з використанням однієї або сімейства споріднених технологій. Web-додаток працює з використанням принципово різних технологій.

Структурні відмінності. Класичний додаток "монолітний". Складається з одного або невеликої кількості модулів. Не використовує сервери БД, Web-сервери і т. д. Web-додаток – "багатокомпонентний". Складається з великої кількості модулів. Обов'язково використовує сервери БД, Web-сервери, сервери додатків.

Відмінності режимів роботи. Класичний додаток працює в режимі реального часу, тобто відомо про дії користувача відразу ж, як тільки вони виконані. Web-додаток працює в режимі "запит-відповідь", тобто відомо про деякий набор дій тільки після запиту на сервер.

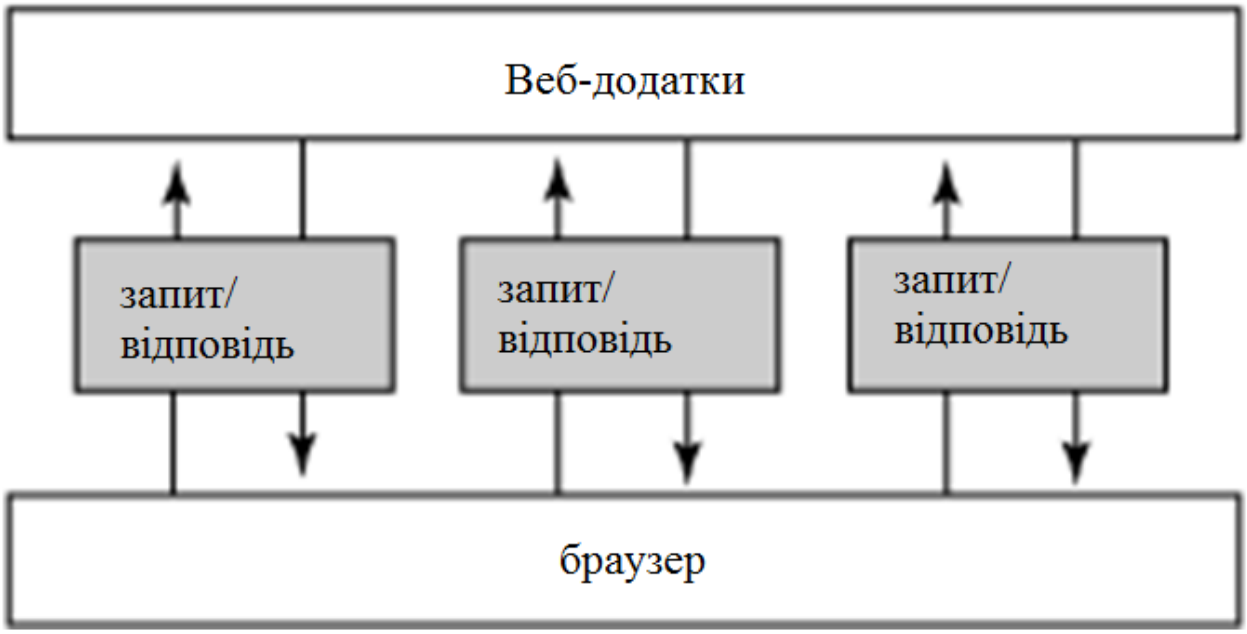


Рисунок 1.5 – Структурна схема режиму роботи Web-додатку.

Відмінності формування інтерфейсу. Класичний додаток використовує для формування інтерфейсу користувача щодо усталені і стандартизовані технології. Web-додаток використовує для формування інтерфейсу технології, які стрімко розвиваються, безліч яких конкурує між собою.

Відмінності роботи з мережею. Класичний додаток практично не використовує мережеві канали передачі даних. Web-додаток активно використовує мережеві канали передачі даних.

Відмінності запуску і зупинки. Класичний додаток запускається і зупиняється рідко. Web-додаток запускається і зупиняється за фактом надходження кожного запиту, тобто дуже часто.

Різниця в кількості користувачів. Класичний додаток: кількість користувачів, які одночасно використовують додаток, піддається контролю, обмежена і є легко прогнозованою [8]. Web-додаток: кількість користувачів, які одночасно використовують додаток, важко прогнозується і може стрибкоподібно змінюватися в широких діапазонах.

Особливості збоїв і відмов. Класичний додаток: вихід з ладу тих чи інших компонентів відразу стає очевидним. Web-додаток: вихід з ладу деяких компонентів надає непередбачуваний вплив на працездатність програми в цілому.

Відмінності в інсталяції. Класичний додаток – процес інсталяції стандартизований і максимально орієнтований на широку аудиторію користувачів. Не вимагає специфічних знань. Додавання компонентів програми виконується стандартним способом з використанням одного і того ж інсталятора. Web-додаток – процес інсталяції часто недоступний кінцевому користувачеві. Інсталяція вимагає специфічних знань. Процес зміни компонент програми не передбачається або потребує кваліфікації користувачів. Інсталятор відсутній [9].

Відмінності в деінсталяції. Класичний додаток: процес деінсталяції стандартизований і виконується автоматично або напівавтоматично. Web-додаток: процес деінсталяції вимагає специфічних знань для втручання адміністратора і часто пов'язаний із зміною коду середовища функціонування програми, БД, налаштування системного ОС.

Особливості середовища функціонування. Класичний додаток: середовище функціонування стандартизовано і не сильно впливає на функціонування програми. Web-додаток: середовище функціонування дуже різноманітне і може мати серйозний вплив на працездатність і серверної, і клієнтської частини.

1.3 Сучасні системи автоматизованого тестування Web-додатків

1.3.1 Apache JMeter

Спочатку додаток JMeter призначався для тестування продуктивності Web-серверів, а тепер перетворився в автоматичний інструмент для тестування з тестовими даними, а також в інструмент для функціонального тестування додатків, файлових серверів, Web-серверів і навіть баз даних. Додаток можна налаштувати так, щоб він симулював N-ну кількість користувачів і потоків, які відвідують певний веб-сервер або додаток. Створюючи симульоване навантаження на Web-додаток, JMeter вимірює його продуктивність. Більш того, є можливість задати кілька повторів з циклами, щоб отримати усереднений результат, реалізувати твердження, а також подивитися результати тестування в графічному і статистичному вигляді.

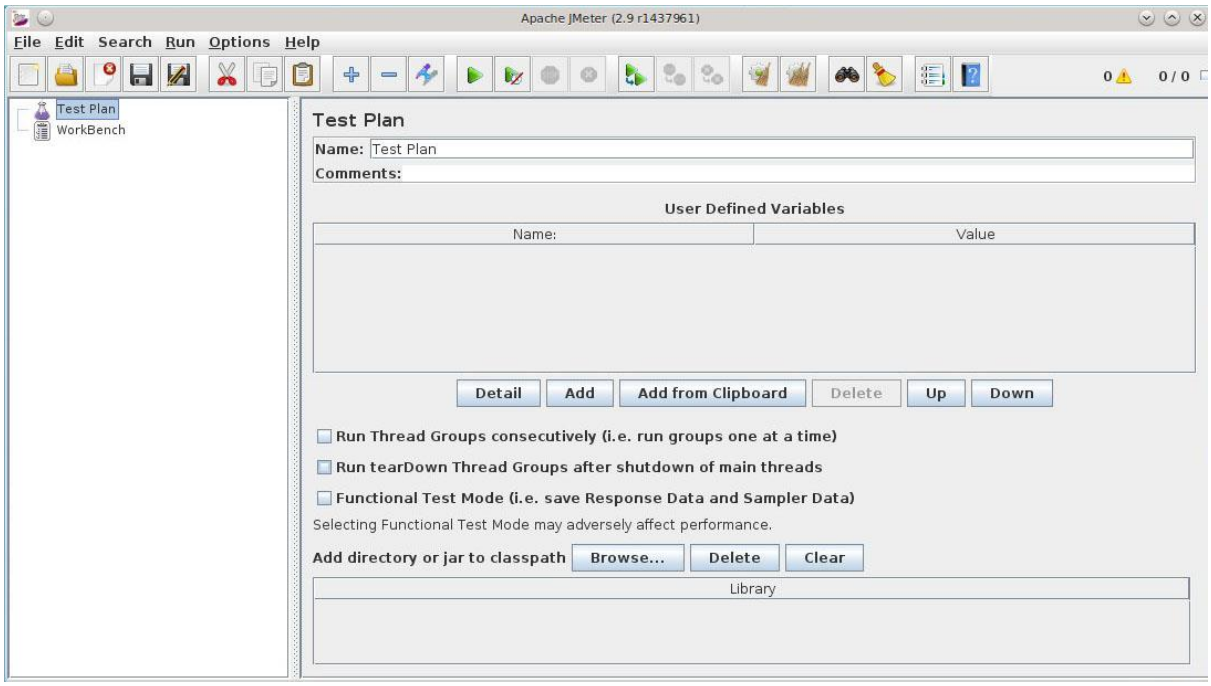


Рисунок 1.6 – Головне вікно JMeter.

Для його роботи потрібно Java на машині, з якої буде проводитися тестування.

1.3.2 Object-Driven Testing

Object Driven Testing (тести, керовані об'єктами) – це такий підхід до автоматизації тестування, при якому тестові скрипти проектуються у вигляді класів, в яких реалізується логіка роботи з додатком. Такі скрипти легше створювати і підтримувати, тому що в тестових випадках використовуються лише методи «високого рівня», дозволяючи приховати подробиці реалізації тих чи інших дій.

У програмному засобу TestComplete є спеціальний елемент проекту ODT, який і дозволяє представити скрипти у вигляді класів з властивостями і методами.

Відразу зауважимо, що в мовах JScript і VBScript ви можете створювати класи, користуючись тільки вбудованими засобами цих мов.

Використання ODT

Структура даних в ODT розділена на класи і дані. Класи містять властивості (які можуть бути як звичайними змінними, так і масивами) та методи (імена методів прив'язуються до існуючих функцій). Дані можу містити звичайні змінні, масиви та об'єкти класів. На відміну від більшості об'єктно-орієнтованих мов, в ODT є цікава особливість: об'єкти класу можуть містити додаткові (власні) методи, які будуть доступні тільки в цьому об'єкті, але не в інших об'єктах цього ж класу.

Як приклад створимо простий клас і об'єкт для калькулятору матеріалів. Двічі натиснемо на елемент «Classes» в Project Explorer, потім у редакторі праворуч натиснемо правою кнопкою миші по елементу «Classes» і виберемо пункт меню New Item. Додасться новий клас «NewClass». Натиснемо клавішу F2 і перейменуємо клас у «Calculator». Зверніть увагу, що тепер у нас стала доступною вкладка Methods. Ця вкладка доступна тільки тоді, коли в редакторі виділено клас або об'єкт класу.

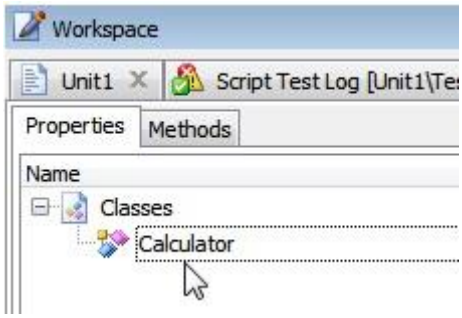


Рисунок 1.7 – Створення класу «Calculator».

Якщо тепер натиснути правою кнопкою миші на імені класу і вибрати пункт меню New Item, то додасться нова властивість:

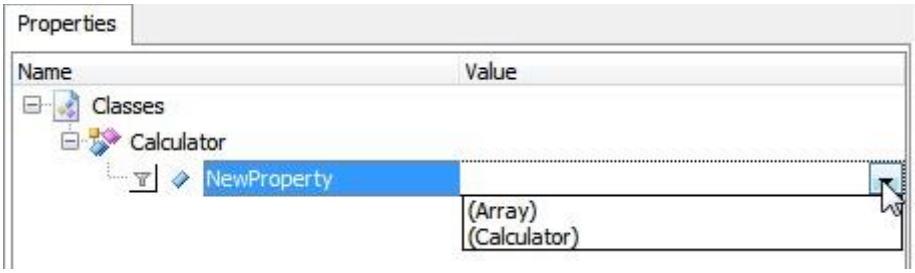


Рисунок 1.8 – Додавання властивості до створенного класу.

У колонку Value можна ввести якесь значення безпосередньо (рядок, число, true / false) або вибрати значення (Array), щоб створити змінну типу масив. Ми назвемо цю властивість «Result» і не будемо зараз присвоювати йому ніякого значення. Ця властивість буде надалі використано для зберігання результату обчислень.

Тепер виділимо в редакторі клас «Calculator» і перейдемо на вкладку Methods. Тут ми визначимо 3 методи (натиснення правої кнопки миши, New Item).

Як вже було сказано, імена методів зв'язуються з існуючими функціями в проекті, для чого на вкладці Methods є колонка Test Procedure. Ми створимо необхідні процедури пізніше, а зараз перейдемо до створення даних (об'єктів).

Для цього в Project Explorer виберемо елемент Data, потім у редакторі даних натиснемо правою кнопкою миші і виберемо пункт меню New item. При

цьому створиться нова група «NewGroup». Дані (або об'єкти) в ODT згруповані по групах, в кожній групі може зберігається скільки завгодно об'єктів. Ми назвемо нашу групу «CalcGroup», після чого додамо в неї новий елемент («Calc») і зв'яжемо його з класом «Calculator», як показано на скріншоті нижче.

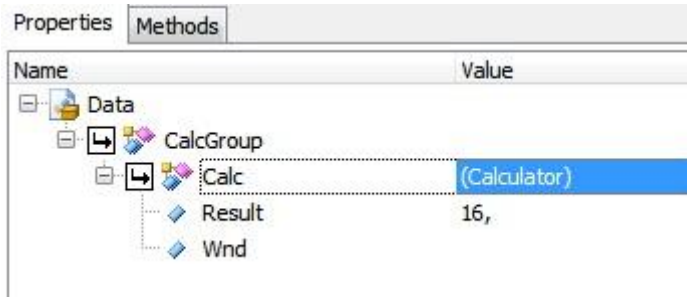


Рисунок 1.9 – Зв’язання об’єкту «Calc» з класом «Calculator».

Зверніть увагу, що як тільки ми зв'язали об'єкт «Calc» з класом «Calculator», ми відразу бачимо його властивість «Result», яке ми визначили для класу.

Додатково визначимо ще одну властивість «Wnd». За допомогою цієї властивості ми будемо звертатися до головного вікна калькулятору матеріалів.

Також на вкладці Methods ми можемо визначити додаткові методи, які будуть доступні для цього об'єкту, але не для класу і не для інших об'єктів цього класу.

Вище ми визначили 3 методи для класу «Calculator» і зараз саме час створити для них тестові процедури. Для цього в будь-якому модулі проекту вставимо наступний код:

```
function _CalcStartODT() {
    TestedApps.Calc.Run();
    var wCalc = Sys.Process("CalcPlus").Window("SciCalc", "Calculator Plus",
1);
    ODT.Data.CalcGroup.Calc.Wnd = wCalc;
    ODT.Data.CalcGroup.Calc.Result = wCalc.Window("Edit", "", 1).WText;
}
```

```
function _CalcStopODT() {
    Sys.Process("CalcPlus").Terminate();
}

function _CalcCalculateODT(expression) {
    var i;
    var wCalc = Sys.Process("CalcPlus").Window("SciCalc", "Calculator Plus",
1);
    for (i = 0; i < expression.length; i + +) {
        wCalc.Keys(expression.substr(i, 1));
    }
    wCalc.Keys("=");
    This.Result = wCalc.Window("Edit", "", 1).WText;
    return This.Result;
}
```

Функція «_CalcStartODT» запускає Калькулятор і привласнює змінній «Wnd» об'єкта «Calc» об'єкт Window("SciCalc"), який є головним вікном калькулятору матеріалів, а також присвоює властивості «Result» значення текстового поля калькулятору матеріалів; функція «_CalcStopODT» закриває Калькулятор, попросту вбиваючи його процес; функція «_CalcCalculateODT» дозволяє розрахувати значення переданого вираження за допомогою калькулятора. Крім того, функція «_CalcCalculateODT» демонструє, як можна за допомогою ключового слова This (або Self у разі DelphiScript) з методу звернутися до власного об'єкту. Зверніть увагу, що в даному випадку ключові слова This і Self необхідно писати з великої літери, оскільки аналогічні слова в нижньому регістрі (this і self) є зарезервованими словами в мовах програмування.

Функції можуть називатися як завгодно, ми ж використовували суфікс «ODT» в їх іменах просто для наочності.

Тепер саме час зв'язати методи класу зі створеними функціями. Для цього повернемося до списку методів створеного раніше класу «Calculator» і виберемо для кожного методу відповідну функцію, як показано нижче на рисунку 1.10.

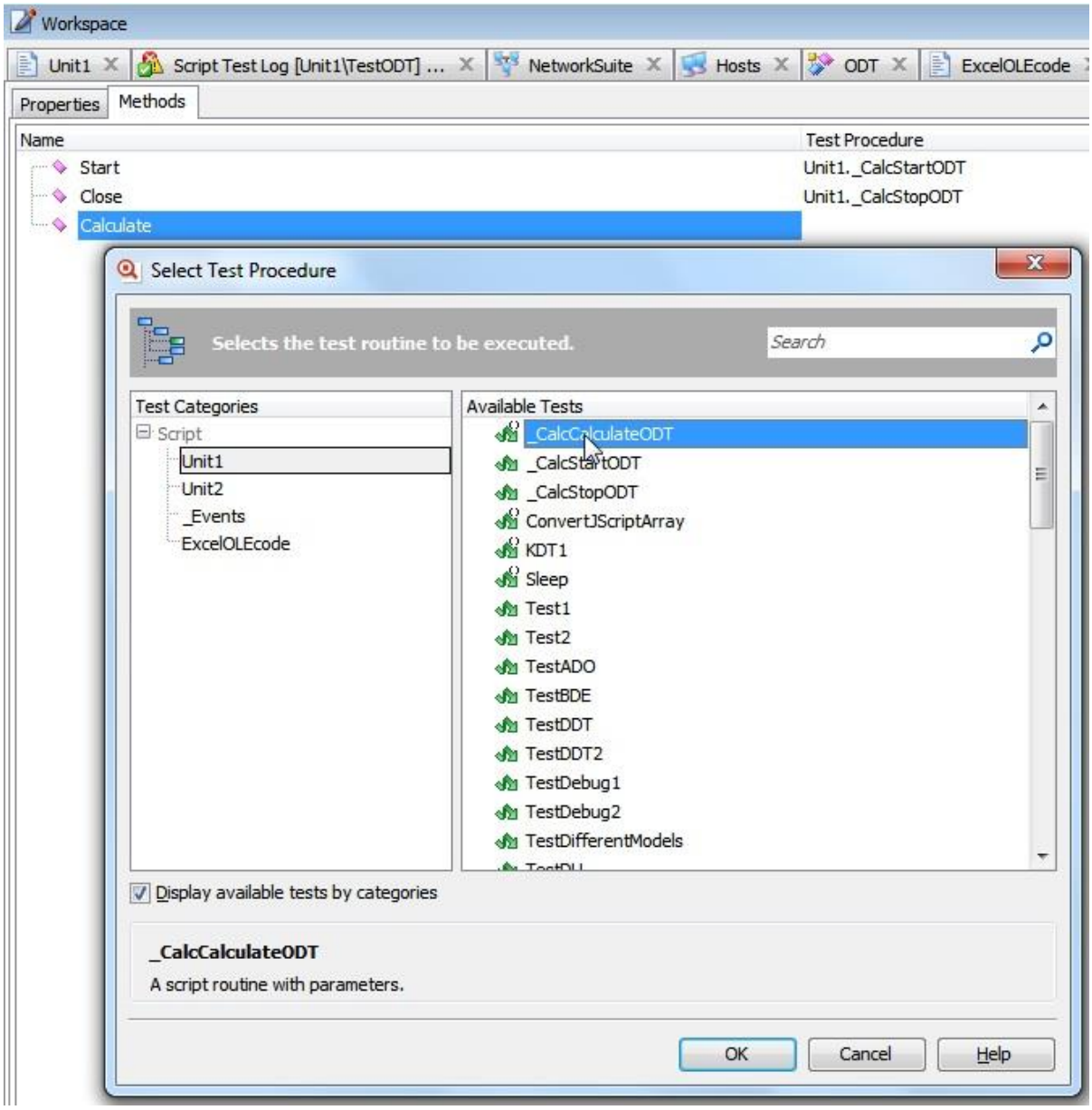


Рисунок 1.10 – Зв'язування методів класу зі створеними функціями

Тепер ми готові написати простий тест, який у Калькуляторі обчислює значення переданого вираження і результат виводить у балку. Код тесту:

```
function TestODT() {
    var calc = ODT.Data.CalcGroup.Calc;
    calc.Start();
    Log.Message(calc.Calculate("(5 +3) * 2"));
    Log.Message(ODT.Classes.Calculator.Result);
    calc.Close();
}
```

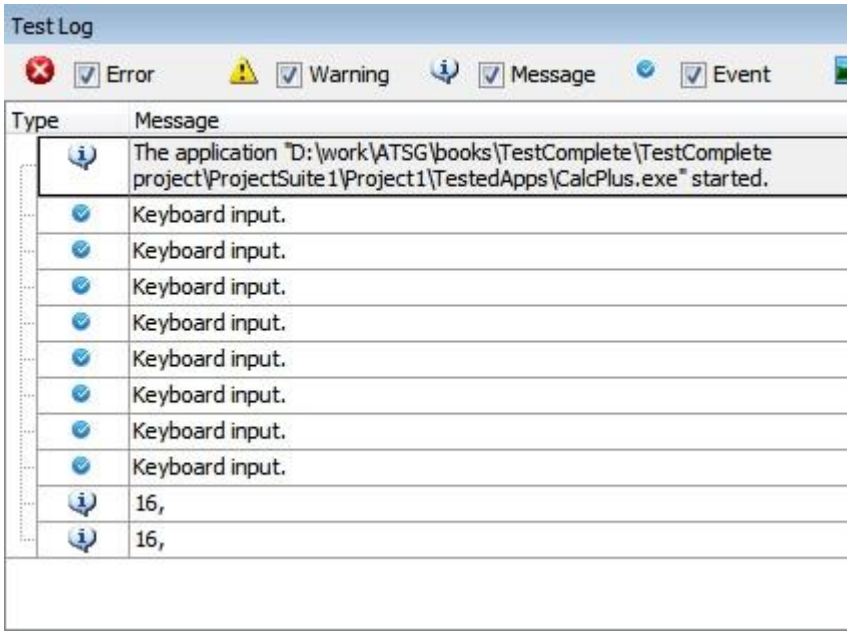


Рисунок 1.11 – Результат роботи програми.

Як бачите, хоча нам і довелося попрацювати, створюючи структуру даних в ODT, відповідні функції та зв'язуючи всі ці дані разом, в результаті ми отримали дуже простий і легкий для читання скрипт, який легко зрозуміти і відредагувати у випадку потреби.

Object Driven Testing - це потужний інструмент, який дозволяє створювати тестові скрипти високого класу.

Крім розглянутих вище можливостей, TestComplete дозволяє створювати класи та об'єкти ODT динамічно (тобто під час роботи скрипта).Якщо вас цікавить ця можливість, зверніться до довідкової системи TestComplete, розділ «Creating Custom Objects Programmatically».

У разі використання JScript або VBScript подібного ефекту можна добитися і за допомогою вбудованих можливостей мов.

У TestComplete є один недолік, пов'язаний з подібним оголошенням класів і методів. Якщо ми оголошуємо звичайну функцію, а потім використовуємо її десь, то ми можемо потім легко перейти до її вихідного коду з будь-якого місця, де використовується ця функція, затиснувши клавішу Ctrl

і натиснути мишею по імені функції. У випадку з класами і методами, проте, такий перехід неможливий. Крім того, з незвички може виявитися важко знаходити в коді місце, де сталася помилка, коли ми двічі клацаємо помилково в балці.

Висновки до розділу

Першим розділом розкрито поняття Web-додатку, його сутність та особливості. Також охарактеризовано принципи тестування Web-додтків та начедено принципові відмінності від тестування програмних додатків. Наведено основні програми для тестування Web-додатків, які детально описано та структуровано. Наступним кроком є висвітлення методологічних аспектів автоматизованого тестування Web-додатків.

РОЗДІЛ 2
МЕТОДОЛОГІЧНІ АСПЕКТИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

2.1 Цільовий аналіз системи автоматизованого тестування Web-додатків

З наведеного аналізу сучасних автоматизованих систем слідує, що для розробки тестів слід звернути особливу увагу на методи які реалізують критерії функціонального тестування, а також розглянути модель предметної області розробки.

Для розробки тестів розглянемо докладніше методи функціональних діаграм і попарного тестування.

Для аналізу предметної області розробимо онтологічну модель.

Функціональна діаграма являє собою формальну мову, на якій транслюється специфікація програми, написана на природній мові. Побудова тестів цим методом здійснюється в кілька етапів [14]:

- вхідні дані предметної області розбиваються на класи – еквівалентність;
- за специфікацією визначаються причини і наслідки, при цьому пі причиною розуміють окреме вхідне значення або клас еквівалентності вхідних даних, а наслідок – це вихідне значення або перетворенні програми (дія, яка вхідна умова надає на ста програми);
- причини і наслідки перетворюється в булевський граф - це і є функціональна діаграма;
- діаграма додається примітками, що задають обмеження і описують комбінації причин і (або) наслідків, які є неможливими через синтаксичні або зовнішні обмеження;

– діаграма перетворюється в таблицю рішень з обмеженими входами, кожен стовпець якої відповідає тесту;
– стовпці таблиці рішень перетворюються в тести.

Базові символи для запису функціональних діаграм показані на рис. 2.1.

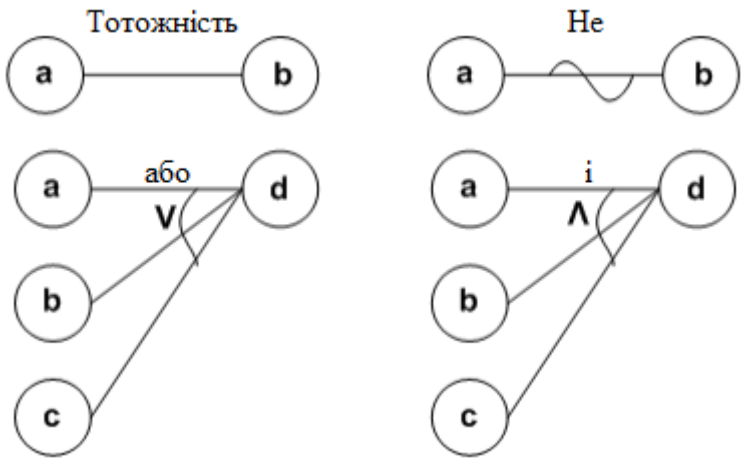


Рисунок 2.1 – Базові символи для запису функціональних діаграм.

При розробці тестів часто доводиться аналізувати роботу системи з великим числом параметрів (наприклад, робота сайту в різних браузерах).

Наприклад, для перевірки всіх поєднань 10 параметрів з 10 значеннями кожен, буде потрібно 10,000,000,000 тестів. Перебрати всі поєднання параметрів неважко. Труднощі полягають в тому, щоб забезпечити при цьому мінімум тестів. Одним з підходів оптимізації кількості тестів є використання ортогональних масивів.

Ортогональний масив [20] – це таблиця $L_m(k^n)$, де m – число рядків, n – число стовпців, яке відповідає числу вхідних параметрів, k – кількість варіантів значень елементів таблиці, і володіє наступними властивостями:

– будь-які два стовпці таблиці містять всі комбінації значень цих стовпців;

– якщо будь-яка пара значень двох стовпців зустрічається кілька разів, то всі можливі парні комбінації значень цих стовпців повинні зустрітися стільки ж разів.

В ортогональних масивах необов'язково всі стовпці повинні мати однакову кількість значень. Існують так звані змішані ортогональні масиви. Наприклад: $L_4(2^3)$- ортогональний масив з чотирма рядками, трьома стовпцями (за кількістю змінних), 2 означає, що всі змінні приймають тільки два значення-1 і 2. $L_{18}(2^1 3^7)$ – змішаний ортогональний масив з вісімнадцятьма рядками, у якого один стовпець зі значеннями 1 і 2, і сім стовпців зі значеннями 1, 2, 3.

Для тестування з використанням ортогональних масивів слід виконати наступні кроки:

– встановити комбінації змінних для вхідних даних;
– визначити значення, які можуть приймати змінні;
– побудувати ортогональний масив, який має стовпець для кожної змінної (можна скористатися програмою STATISTICA);
– поставити у відповідність кожному тестовому випадку комбінацію значень змінних, розташованих у рядку побудованого масиву.

Наприклад, матриця, в стовпцях якої може бути значення 1 або 2, містить всі можливі комбінації трьох цифр (наведено нижче у рисунку 2.2):

| 1 | 1 | 1 |

| 2 | 1 | 1 |

| 1 | 2 | 1 |

| 1 | 1 | 2 |

| 2 | 2 | 1 |

| 1 | 2 | 2 |

| 2 | 1 | 2 |

| 2 | 2 | 2 |

Рисунок 2.2 – Вихідна матриця.

Ортогональний масив, отриманий по вихідній матриці наступний (рис. 2.3).

| 1 | 1 | 1 |

| 1 | 2 | 2 |

| 2 | 1 | 2 |

Рисунок 2.3 – Ортогональний масив.

В математиці, для ортогонального масиву, чиї записи складаються з фіксованого кінцевого набору символів (як правило, {1,2,...,n}), існує ціле число t таке, що для кожної множини з T стовпців таблиці, всі можливі T-кортежі з вихідних символів, утворені записом в кожному рядку символів цих стовпців, з'являються однакову кількість разів. Число T називається силою ортогональності. Задля розглянутого прикладу (рис. 2.2 та рис. 2.3) ортогональний масив з безліччю символів {1,2} має силу – 2. При цьому чотири впорядковані пари, утворені першою і третьою колонках, а саме (1,1), (2,1), (1,2) і (2,2), породжують всі можливі впорядковані пари з двох елементів набору, і кожен з'являється рівно один раз. Другий і третій стовпці дають: (1,1), (2,1), (2,2) і (1,2). І в цьому випадку також з'являються всі можливі

впорядковані пари. Аналогічно можна отримати всі можливі впорядковані пари, розглядаючи перший і другий стовпці.

Протягом багатьох років, був розроблений цілий ряд комбінаторних стратегій для того, щоб допомогти тестувальникам вибрати таку підмножину вхідних комбінацій, яка дозволила б максимально збільшити імовірність виявлення дефектів: вибіркове тестування, «кожен-вибір» ((each-choice) і «підстава вибору» (base choice), анти рандомізація ((antirandom), стратегія тестування t-способами (twice testing strategies) і інший. Парне тестування (pairwise testing) [23] є найбільш видним серед них.

Метод парного тестування заснований на досить простій ідеї, що переважна більшість помилок в All-Pairs Algorithm (алгоритм всіх пар – це комбінаторна методика, яка була спеціально створена для парного тестування. В її основі лежить вибір можливих комбінацій значень всіх змінних, в яких містяться всі можливі значення для кожної пари змінних. Виходячи із визначення, число комбінацій буде менше, ніж при використанні ортогональних масивів.

Для тестування з використанням All-Pairs алгоритму виконують наступні кроки:

- аналогічно, як для ортогональних масивів, визначають таблицю всіх змінних і їх значень;
- залишають в таблиці тільки всі можливі унікальні комбінації пар значень змінних.

Розглянемо приклад формування тестів в техніці парного тестування. Інтерфейс користувача програми містить список з 10 елементами (скажімо, 0,1,2,3,4,5,6,7,8,9) разом з прапорцем, радіо-кнопкою, текстовим полем і кнопкою «ОК». Текстове поле може приймати значення тільки в діапазоні від 1 до 100. Нижче наведено значення, які кожен з об'єктів графічного інтерфейсу може приймати:

Список Box – 0,1,2,3,4,5,6,7,8,9

Check Box – зареєстрований або незареєстрований

Radio Button – ON або OFF

Text Box – будь-яке значення від 1 до 100

Вичерпна кількість тестів для програми обчислюється наступний чином:

List Box = 10;

Check Box = 2;

Radio Button = 2;

Text Box = 100;

Загальне число тестів: 10*2*2*100 = 4000

Загальне число негативних тестів > 4000.

Тепер, ідея розробки тестів полягає в тому, щоб зменшити кількість тестів. Спочатку з'ясуємо число випадків з використанням звичайного методу тестування програмного забезпечення. Є можливість розглядати значення List Box як 0 та інші. Число значень радіо-кнопки і Check Box не можуть бути зменшені, так що кожен з них буде мати 2 комбінації (включений або вимкнений). Значення текстового поля може бути зменшено до трьох: Допустиме ціле число, неприпустиме ціле (Invalid Integer), альфа – спеціальний символ.

Таким чином, число випадків з використанням програмного забезпечення в методиці тестування складе: 2 * 2 * 2 * 3 = 24 (в тому числі негативних випадків).

Далі зменшимо кількість поєднань значень параметрів в техніці парного теестування:

– виберемо параметри таким чином, що параметр з найбільшою кількістю значень був першим і з найменшою – останнім;

- заповнюємо таблицю по стовпцях. List Box може приймати 3 значення;
- у наступній колонці буде Text Box (може приймати 2 значення);
- перевіряємо, що покриті всі комбінації між списком і текстовим полем;
- будемо використовувати ту ж стратегію для перевірки радіо – кнопки і Check Box.
- перевіряємо, що всі значення пар параметрів покриті (таблиця 2.1).

Таблиця 2.1 – Покриття тестових значень.

TextBox	List Box	Check Box	Перемикач
Int	0	перевірити	ON
Int	інші	зніміть прапорець	OFF
Invalid Int	0	перевірити	ON
Invalid Int	інші	зніміть прапорець	OFF

Для парного тестування використовуються алгоритми, засновані на побудові ортогональних масивів або на All-Pairs алгоритмі, які спираються на теоретичні дослідження в області комбінаторних алгоритмів, алгоритмів дискретної математики і, зокрема, латинських квадратів [14-20].

Зазвичай на практиці використовують деякі програмні реалізації цього алгоритму, так як вручну перебирати всі пари досить трудомістко.

Використовуємо онтологічний підхід [21] для аналізу предметної області розробки тестів. Спочатку розглянемо параметри тестового випадку.

Під тестовим випадком (test case) зазвичай розуміється структура виду:

$$Action > Expected\ Result > Test\ Result$$

У таблиці 2.2 показаний приклад тестового випадку.

Таблиця 2.2 – Приклад тестового випадку.

Action Дія	Expecred Result Очікуваний результат	Test Result Результати тесту (passed/failed/blocked)
Відкрити сторінку "Login"	Сторінка "Login" відкривається	Passed

Опис тестового випадку може включати наступні частини (таблиця 2.3, 2.4) [17].

Таблиця 2.3 – Опис тестового випадку (варіант 1).

Pre conditions	Список дій або умов, які призводять систему до стану придатного для проведення тестування
Test case description	Список дій для отримання результату, на підставі якого можна зробити висновок про те, чи задовольняє дана реалізація програми, поставленим вимогам
Post conditions	Список дій, що переводять систему в первісний стан (стан до проведення тесту)

Таблиця 2.4 – Опис тестового випадку (варіант 2).

ID	Номер тестового випадку. Служить для унікальної ідентифікації тесту серед інших тестових випадків
Test Case Priority	Пріоритет. Вимірюється від 1 до N: 1 – найвищий: N – найнижчий (для не дуже великих проектів раціонально вибирати n=4)
Summary	Короткий опис проблеми: що сталося і при яких умовах програма працює не вірно
Steps	Опис кроків для відтворення помилки
Expected Result	Опис очікуваного результату після виконання конкретних кроків тесту
Pass/Fail	Статус тестового випадку після виконання: якщо очікуваний результат збігається з реальним, то – Pass, в іншому випадку – Fail

Описи параметрів тестових випадків необхідні для формування звітів, що містять інформацію про виявлені проблеми.

Простіша модель онтології (без словника і без визначення типу відносин) наведена нижче. Під онтологією в цій роботі розуміється упорядкована трійка виду [9 – 11]:

$$O = \langle C, R, F \rangle,$$

Де:

- C - кінцева безліч концептів (понять, термінів) предметної області, яку представляє онтологія O;
- R - кінцева безліч відносин між концептами заданої предметної області;
- F - кінцева безліч функцій інтерпретації (аксіоматизації), заданих на концептах та/або відносинах онтологій O.

Природним обмеженням, що накладається на безліч C, є його кінцівка і не порожнеча.

Онтологічна модель тестового випадку наведена у **додатку А** (див. лист 73).

2.2 Алгоритм реалізації системи автоматизованого тестування Web-додатків

Поетапний алгоритм реалізації системи автоматизованого тестування Web-додатків складається з таких пунктів:

Підготовча фаза тестування сайту. спеціаліст–тестувальник на підставі отриманої документації, складає тест-план (Test Plan).

Функціональне тестування. Сама трудомістка частина випробування ресурсу. На цьому етапі тестуються всі функціональні вимоги програмного продукту, робота посилань, пошук неробочих гіперпосилань. Йде перевірка підвантаження файлів на сервер, роботи лічильників на сторінках порталу, користувацьких форм (наприклад, контакти, зворотний зв'язок, підписка, додавання повідомлень і т. д.). Перевіряється, чи відповідає вміст сторінок сайту, що є базовим.

Тестування верстки. При випробуванні верстки перевіряємо в суворій послідовності:

- розташування елементів, чи відповідають вони своїм макетам,
- оптимізацію графічних зображень,
- валідність коду,
- кросбраузерість (як працює в різних браузерах).

Перевірка зручності користування (або Usability Testing). Usability тестування грунтується на залученні в якості тестувальників користувачів, аналізуються всі результати і думки.

Тестування безпеки. На цьому етапі тестувальник займається перевіркою захисту всіх закритих сторінок.

Перевірка продуктивності. Завдання – визначити швидкість роботи сайту при заданому навантаженні. Тут застосовують навантажувальне тестування (Load Testing) і тестування швидкодії.

Робота над помилками. Звіт про знайдені помилки тестувальник надає розробникам сайту. Складається графік виправлення неполадок і проводиться повторне тестування.

2.3 Вибір інструментарію

Для вибору інструментарію автоматизованого тестування по-перше слід звернути увагу наскільки добре інструмент для автоматизації розпізнає елементи управління в додатку. У разі коли елементи не розпізнаються варто пошукати плагін, або відповідний модуль. Якщо такого немає – від інструменту краще відмовитися. Чим більше елементів може розпізнати інструмент – тим більше часу можна заощадити на написанні і підтримки скриптів.

По-друге потрібно звернути увагу на те скільки часу потрібно на підтримку скриптів написаних за допомогою обраного інструменту. Для цього необхідний простий скрипт, який вибирає пункт меню, а потім уявіть, що змінився пункт меню який необхідно вибрати. Якщо для відновлення працездатності сценарію доведеться перезаписати скрипт цілком, то інструмент не оптимальний, так як реальні сценарії набагато складніше. Найкраще той інструмент, який дозволяє винести назву кнопки в змінну на початку скрипта і швидко замінити її значення.

По-третє, наскільки зручний інструмент для написання нових скриптів. Скільки потрібно на це часу, наскільки можна структурувати код (підтримка ООП), наскільки код читаємо, наскільки зручне середовище розробки для рефакторинга (переробки коду) і т.п.

Найкраще для прийняття правильного рішення про автоматизацію відповідати на питання «Навіщо? Що? Як?» Саме в такому порядку. Це допоможе уникнути даремно витрачений час і фінанси. З іншого боку можна

отримати надійність, швидкість і якість.

Для розробки функціональних тестів і тестування Web-додатку були використані наступні інструменти: Selenium Web Driver, Maven, Intellij IDEA, Selenium IDE 2.9.1.

Інструмент для автоматизованого тестування – це програмне забезпечення, за допомогою якого здійснюється створення, налагодження, виконання і аналіз результатів прогону тест-скриптів (Test Scripts – це набори інструкцій для автоматичної перевірки певної частини програмного забезпечення).

Коротко розглянемо такі інструменти Selenium Web Driver, Maven, Intellij IDEA, TestNG для автоматичного тестування.

Selenium Web Driver, або просто Web Driver – це драйвер браузера, тобто не має призначеного для користувача інтерфейсу, програмна бібліотека, що дозволяє іншим програмам взаємодіяти з браузером: управляти його поведінкою, отримувати від браузера якісь дані і змушувати браузер виконувати якісь команди.

Apache Maven – це інструмент для збірки Java проекту: компіляції, створення jar, створення дистрибутива програми та документації.

Прості проекти можна зібрати в командному рядку. Якщо збирати великі проекти з командного рядка, то команда для збірки буде дуже довгою, тому її іноді записують в bat / sh скрипт. Але такі скрипти залежать від платформи. Для того щоб позбутися від цієї залежності і спростити написання скрипта використовують інструменти для збірки проекту.

IntelliJ IDEA – інтегроване середовище розробки програмного забезпечення на багатьох мовах програмування, зокрема Java, Go, JavaScript, Python.

TestNG – фреймворк для тестування, написаний на Java, багато успадкувавши від існуючої функціональності JUnit та NUnit; має нові інноваційні функції, які роблять його потужним і простим у використанні.

Jenkins – проект для безперервної інтеграції з відкритим вихідним кодом, написаний на Java.

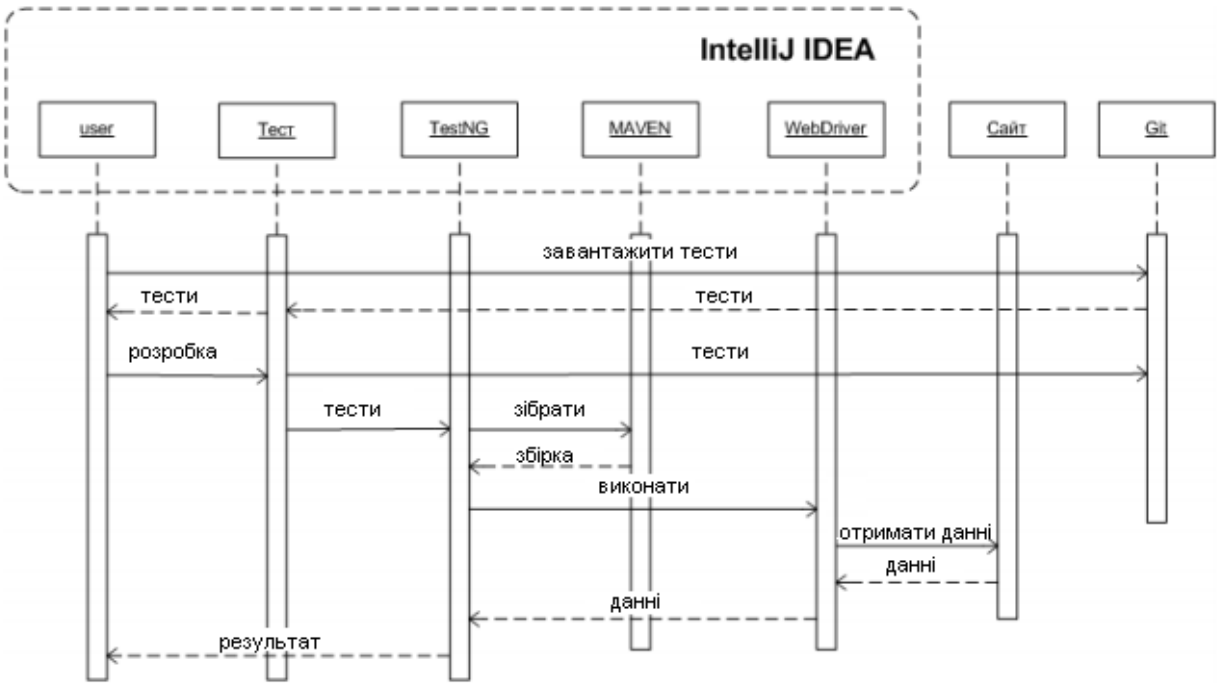


Рисунок 2.8 – Взаємодія інструментів автотестування.

У якості додаткового інструментарію використовуємо PICT - вільний інструмент, розроблений Microsoft, для парного тестування та All-Pairs.

Формування тестових наборів даних відбувається таким чином, що кожне тестоване значення кожного з параметрів, що перевіряються хоча б один раз поєднується з кожним тестованим значенням всіх інших параметрів, що перевіряються.

Такий підхід приблизно і становить суть техніки Pairwise Testing (парне тестування) - ми не перевіряємо всі поєднання всіх значень, але перевіряємо всі пари значень. Зрозуміло, інструмент PICT, як і техніку парного тестування застосовувати саме в тих випадках, в яких це доцільно. От якщо, наприклад, у нас є форма з якоюсь кількістю різних полів, дані з яких просто зберігаються в базі даних, то застосування цієї техніки практично не має сенсу, адже дані не взаємодіють один з одним (хоча і можуть бути нюанси, як ці дані зберігаються

в базу). А парне тестування - це та техніка, застосовувати яку варто саме в разі взаємодіючих значень (для невзаємодіючих найчастіше досить просто окремої перевірки кожного з параметрів). Під взаємодіючими параметрами я розумію в першу чергу ті, які впливають на результат не просто своїми власними окремими значеннями, але саме комбінаціями один з одним. До речі, ось хороший приклад взаємодії-визначення варіантів тестового середовища, наприклад, з декількох операційних систем, браузерів і дозволів монітора. Отже, якщо параметри завдання взаємодіють, то, здавалося б, тут парне тестування і стане тим чудовим рішенням, яке і сили заощадить, і повне тестування забезпечить. Але звичайно, ця техніка забезпечує досить високе покриття, але далеко не повне – якщо раптом помилка виникає при поєднанні трьох, чотирьох або більше параметрів, то парне тестування може і не допомогти. По-хорошому, застосуванню парного тестування повинен передувати аналіз тестованого додатка на предмет того, наскільки тестування саме поєднань пар параметрів є для нього доцільним.

All-Pairs Algorithm (алгоритм всіх пар) – це комбінаторна методика, яка була спеціально створена для попарного тестування. В її основі лежить вибір можливих комбінацій значень всіх змінних, в яких містяться всі можливі значення для кожної пари змінних. Виходячи з визначення при цьому буде отримано менше число комбінацій, чим при використанні ортогональних масивів.

Для тестування з використанням All-Pairs алгоритму виконують наступні кроки:

- аналогічно, як для ортогональних масивів, визначають таблицю всіх змінних і їх значень;
- залишають в таблиці тільки всі можливі унікальні комбінації пар значень змінних;

Зазвичай на практиці використовують деякі програмні реалізації цього алгоритму, так як вручну перебирати всі пари досить трудомістко.

Розглянемо ще одну задачу з області тестування. Нехай потрібно перевірити працездатність Web-сайту в різних конфігураціях браузерів, операційних системах і на платформах з різною розрядністю. Наприклад, задані чотири конкретні операційні системи: Windows XP SP 3, Windows 7 SP 1, Debian 7.1, Ubuntu 12.04, для двох разрядностей: x86, x64, і три браузера: Chrome, Firefox, Safari.

При цьому отримуємо: 4 x 2 x 3 = 24 конфігурації для тестових наборів, які потрібно перевірити в загальному випадку. Можна зменшити кількість тестових наборів за допомогою AllPairs.

Вхідні дані для модуля AllPairs можуть бути задані як список списків рядків, значення яких однозначно характеризують параметри для тестових наборів. Для нашої задачі це можуть бути просто назви параметрів для конфігурацій:

- список можливих ОС:Windows XP SP 3, Windows 7 SP 1, Debian 7.1, Ubuntu 12.04;
- список розрядностей: x86, x64;
- список браузерів: Chrome, Firefox, Safari.

Завдання вхідних параметрів у Python-коді і використання AllPairs для побудови їх комбінацій може виглядати як в прикладі нижче (передбачається, що каталог ext, що містить пакет metacomm, вже лежить поруч зі скриптом allpairs.py). Код до скрипта allpairs.py наведено у **додатку Б** (див. лист 67).

В скрипті кожен новий рядок змінної inputData містить список допустимих значень для окремого параметра в тестованих конфігураціях.

Після визначення таким чином вхідних даних, потрібно просто виконати скрипт командою:

python allpairs.py

На виході буде отримано пронумерований список рядків виду:

Combination 0: ['Windows XP SP 3', 'x86', 'chrome'] ...

Як бачимо, комбінацій всього 12 - в два рази менше, ніж їх повне число в загальному випадку. А якщо застосувати цей алгоритм для прикладу, то замість 1024 повних комбінацій, отримаємо всього 8!

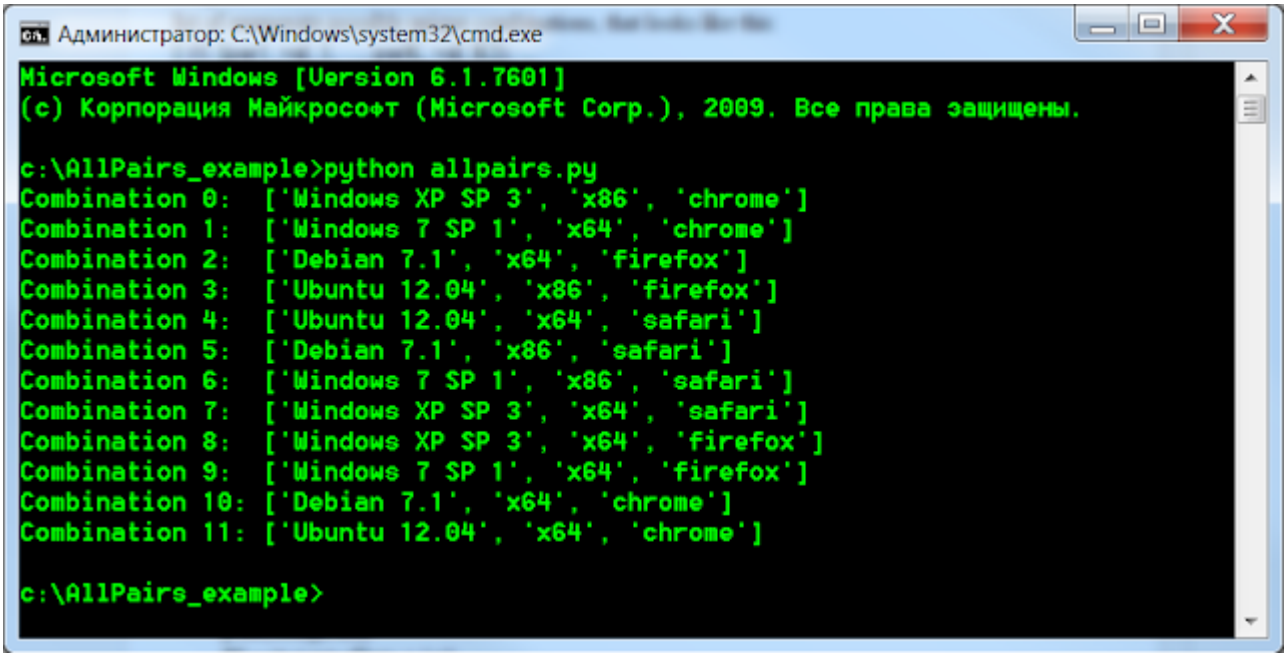


Рисунок 2.9 – Приклад генерації комбінацій за допомогою алгоритму AllPairs.

Перевизначаючи в скрипті allpairs.py значення параметра inputData аналогічним чином, після його відпрацювання можна отримати оптимальні комбінації тестових наборів для всіх подібних завдань. А з метою подальшої автоматизації тестування можна використовувати дані, що генеруються, наприклад, для запуску автотестів з потрібними параметрами конфігурації.

Висновки до розділу

У рамках другого розділу описано цільовий аналіз системи автоматизованого тестування Web-додатків, наведено алгоритми системи автоматизованого тестування Web-додатків. Здійснено вибір інструментарію для проведення подальших досліджень. У якості додаткового інструментарію використовуємо PICT, це програма від Microsoft та All-Pairs.

Наступним кроком є проведення практичного дослідження за темою, розробка та тестування системи автоматизованого тестування Web-додатків.

РОЗДІЛ 3
ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

3.1 Розробка системи автоматизованого тестування Web-додатків

У рамках дипломної роботи розглянемо приклад розробки тестів реєстрації користувачів сайту.

Для спрощення моделі тесту будемо вважати, що користувачеві необхідно вказати логін, пароль, адресу електронної пошти і встановити прапорець згоди на використання його даних. В якості значень параметрів реєстрації приймемо: правильне значення (параметр заданий і має допустиме, вірне значення) і неправильне значення (параметр не заданий або має неприпустиме значення). Функціональна діаграма для розробки тестів реєстрації наведена на рисунку 3.1.

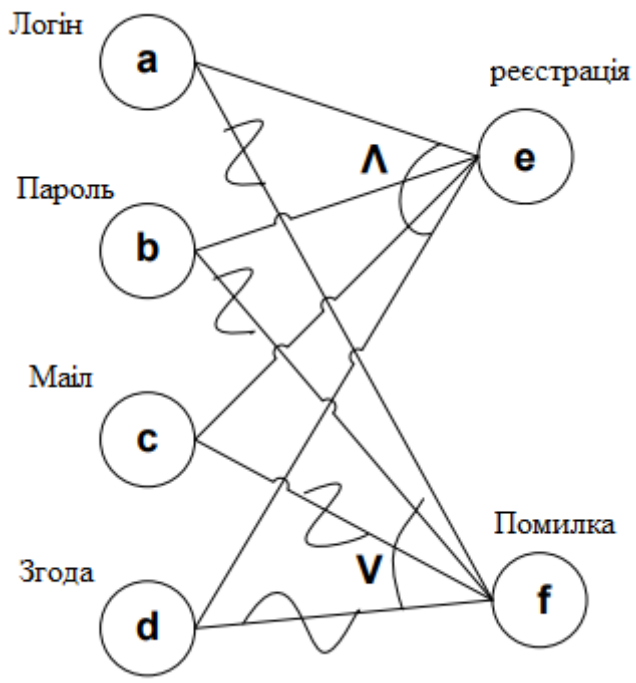


Рисунок 3.1 – Функціональна діаграма реєстрації користувача сайту.

Розробимо програму, яка обчислює всі можливі комбінації значень параметрів і будує по ним таблицю рішень. В основі формування комбінації значень параметрів використовуємо рекурсивний алгоритм, для опису діаграми (булевого графа) будемо використовувати динамічну структуру ступінчастого масиву. Скріншот роботи програми для формування таблиці рішень показаний на рисунку 3.2.

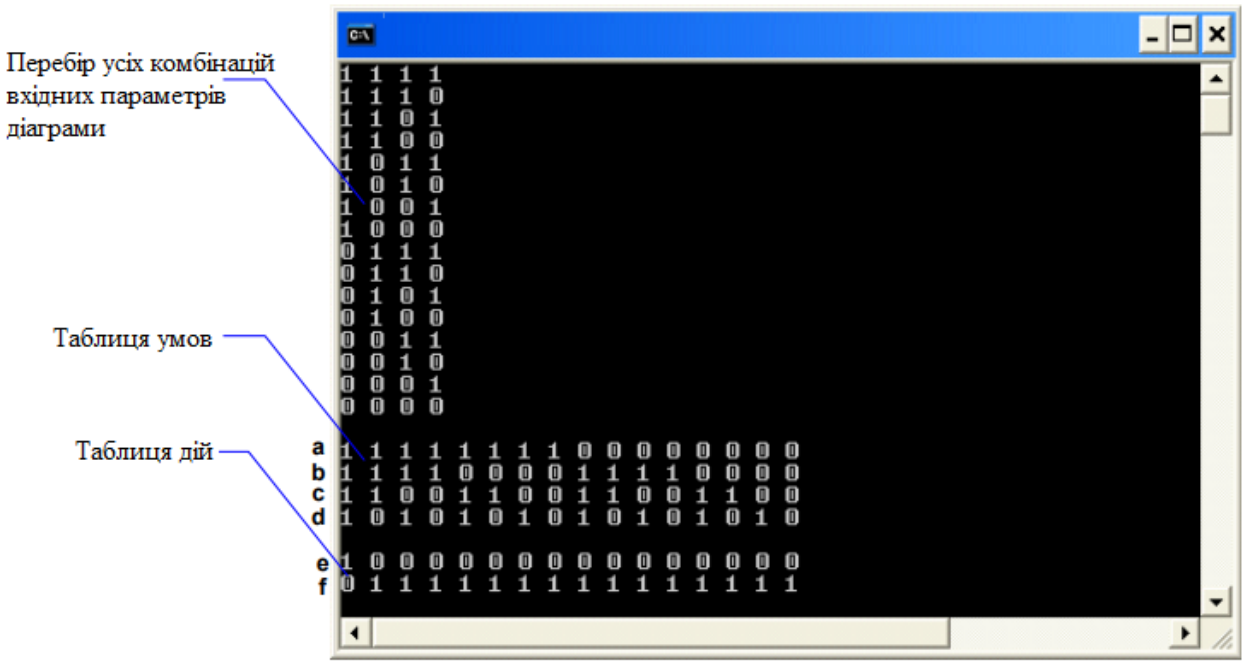


Рисунок 3.2 – Результати роботи програми побудови таблиці рішень по функціональній діаграмі.

Далі проведемо формування тестів на основі методів парного тестування.

3.2 Тестування системи автоматизованого тестування Web-додатків

PICT (Pairwise Independent Combinatorial Testing) – вільний інструмент, розроблений Microsoft, для парного тестування [16].

PICT досить зручний інструмент для швидкого створення набору комбінацій тестових даних, особливо при наявності великої кількості не сильно пов'язаних між собою параметрів (що дозволяє не проводити тестування всіх можливих варіантів). Однак потрібно ретельно створити модель, щоб тестове покриття було задовільним.

Інструмент PICT, як і техніку попарного тестування, доцільно застосовувати у разі взаємодіючих значень (для невзаємодіючих найчастіше досить окремої перевірки кожного з параметрів). Під взаємодіючими параметрами розуміються ті, які впливають на результат не просто своїми власними окремими значеннями, але саме комбінаціями один з одним (наприклад, визначення варіантів тестового середовища, наприклад, з декількох операційних систем, браузерів і дозволів монітор.)

На рисунках 3.3-3.6 показано використання програми PICT для створення тестів реєстрації користувача. У цьому випадку необхідно створити комбінації наступних параметрів інтерфейсу користувача: логін, пароль, E-mail, прапорець згоди.

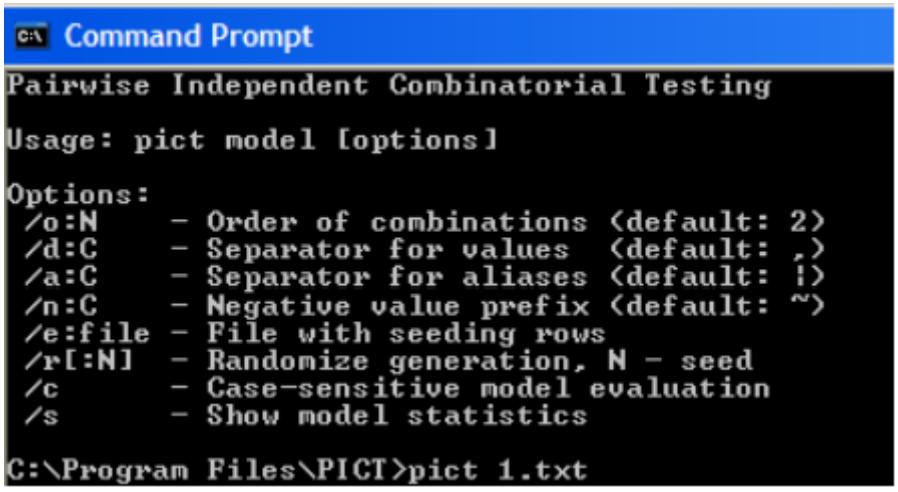


Рисунок 3.3 – Налаштування програми PICT.

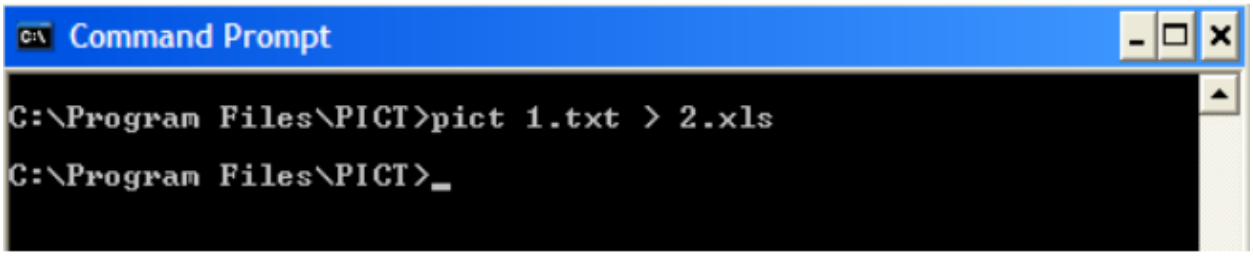


Рисунок 3.4 – Запуск програми PICT.

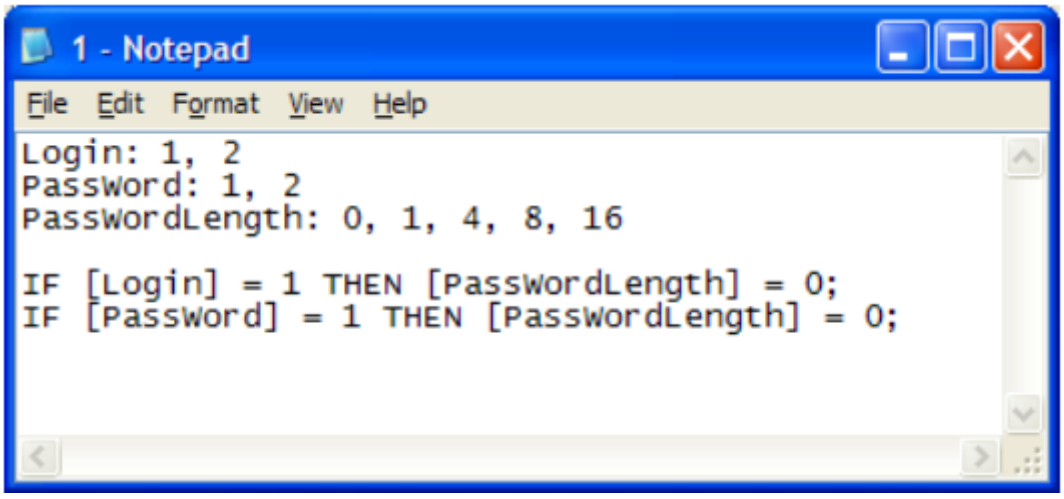


Рисунок 3.5 – Вихідний текстовий файл програми PICT.

	A	B	C
1	Login	PassWord	PassWordLength
2	2	2	4
3	2	2	1
4	1	2	0
5	2	2	16
6	2	1	0
7	2	2	8
8	1	1	0

Рисунок 3.6 – Результуючий файл програми PICT.

Розглянемо приклад перевірки реєстрації користувача на сайті AllPairs з використанням в скриптах генератора комбінацій AllPairs, розробленого MetaCommunications Engineering для Python і такого, що реалізує однойменну методику.

Вхідні дані для модуля AllPairs можуть бути задані як список списків з рядків, значення яких однозначно характеризують параметри для тестових наборів. Для даного завдання:

- логін: GoodLogin, NoLogin;
- пароль: GoodPswd, NoPswd;
- електронна пошта: GoodEmail, NoEmail;
- погоджувальний прапорець: CheckAgree, NoAgree.

Завдання вхідних параметрів в Python-коді і використання AllPairs для побудови їх комбінацій може виглядати так (передбачається, що каталог ext, що містить пакет metacomm, вже лежить поруч зі скриптом allpairs.py). Код до скрипта allpairs.py наведено у **додатку Б** (див. лист 67).

У скрипті кожен новий рядок змінної «inputData» містить список допустимих значень для окремого параметра в тестованих конфігураціях. Після визначення вхідних даних, потрібно просто виконати скрипт командою:

python allpairs.py

На виході буде отримано пронумерований список рядків виду:

Combination 0: ['GoodLogin', ' GoodPswd', ' GoodEmail', 'GoodAgreen'].

В якості вхідних даних для програми All-Pairs можна використовувати .txt-файл з таблицею параметрів, стовпці якої розділені табуляцією.

Маючи вихідний файл необхідно запустити консоль і набрати там рядок виду:

C:\allpairs.exe C:\input.txt > C:\re.txt

де:

C:\allpairs.exe – повний шлях до додатка allpairs.exe

C:\input.txt – шлях до вихідного файлу з таблицею параметрів

C:\re.txt – шлях і ім'я файлу, який буде створений в результаті роботи програми.

Результуючий файл буде містити в собі готовий перелік перевірки.

Застосуємо дану методику для розробки тестів реєстрації користувача. Вхідна інформація, запуск програми та результати наведено на рисунках 3.7-3.9.

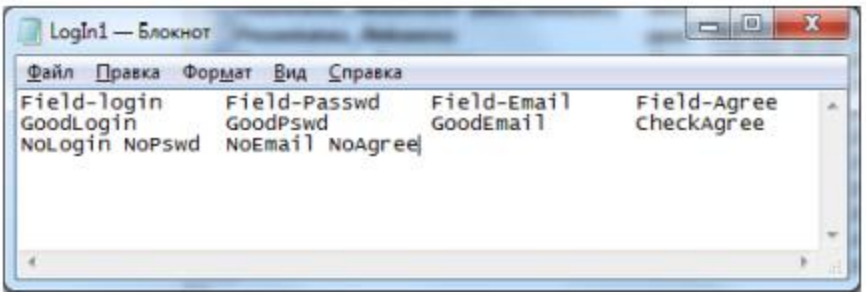


Рисунок 3.7 – Вихідний файл.

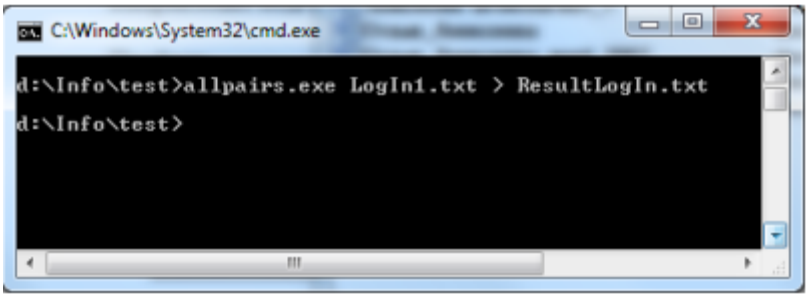


Рисунок 3.8 – Запуск програми.

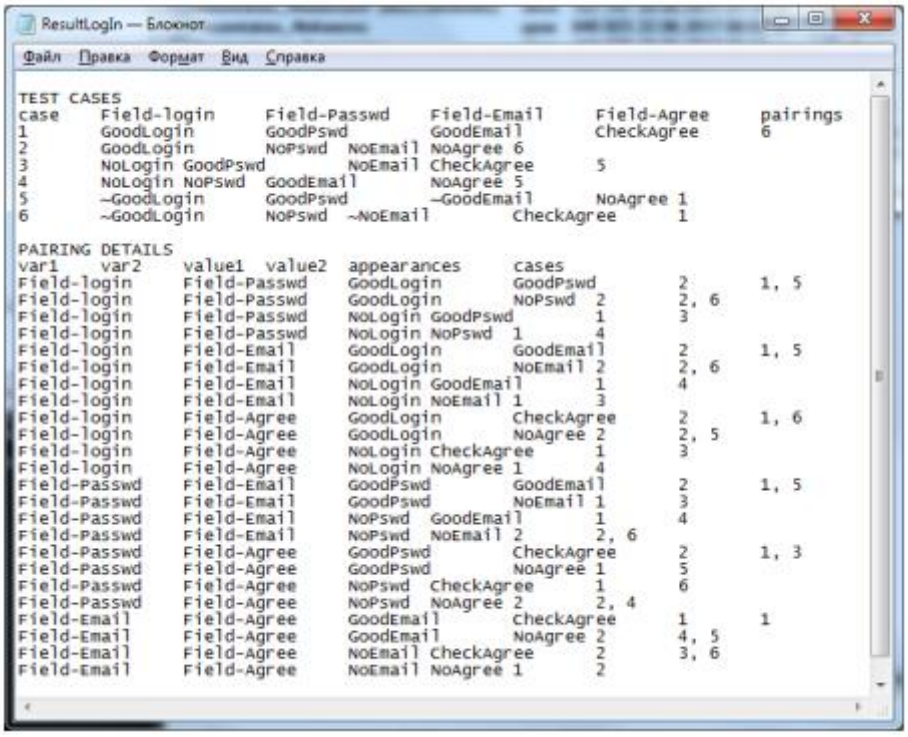


Рисунок 3.9 – Результат.

Pairings – кількість унікальних пар в тестовому випадку, Pairing details – перелік всіх пар всіх параметрів, Appearances – кількість разів, скільки та або інша пара фігурує в випадках, Cases – номери випадків, де фігурує дана пара.

3.3 Аналіз отриманих результатів

На рисунку 3.10 представлена узагальнена структура системи автоматизації тестування, в якій створюється і зберігається наступна інформація:

- набір тестів, достатній для покриття тестованої програми відповідно до обраного критерію тестування – як результат ручної або автоматичної розробки (генерації) тестових наборів і драйвер / монітор пропуску тестового набору;
- результати прогону тестового набору, зафіксовані в Log-файлі. Log-файл містить траси (протоколи), що представляють собою реалізовані при тестовому прогоні послідовності деяких подій (значень окремих змінних або їх сукупностей) і точки реалізації цих подій на графі програми. У складі трас можуть бути присутніми послідовності явно і неявно заданих міток, які задають шлях реалізації трас на керуючому графові програми, сукупності значень змінних на цих позначках, величини проміжних результатів, досягнутих на деяких мітках і т.п.;
- статистика тестового циклу, що містить: результати пропуску кожного тесту з тестового набору і їх порівняння з еталонними величинами;
- факти, що послужили підставою для прийняття рішення про продовження або закінчення тестування;
- критерій покриття і ступінь його задоволення, досягнута в циклі тестування.

Результатом аналізу кожного прогону є список проблем, у вигляді помилок і дефектів, який заноситься в базу розвитку проекту. Далі відбувається робота над помилками, де кожна піднята проблема ідентифікується, відноситься до відповідного модулю і розробнику, що

забезпечує гарантію її рішення (виправлення або віднесення до списку відомих проблем, вирішення яких з тих чи інших причин відкладається) в наступних збірках програми.

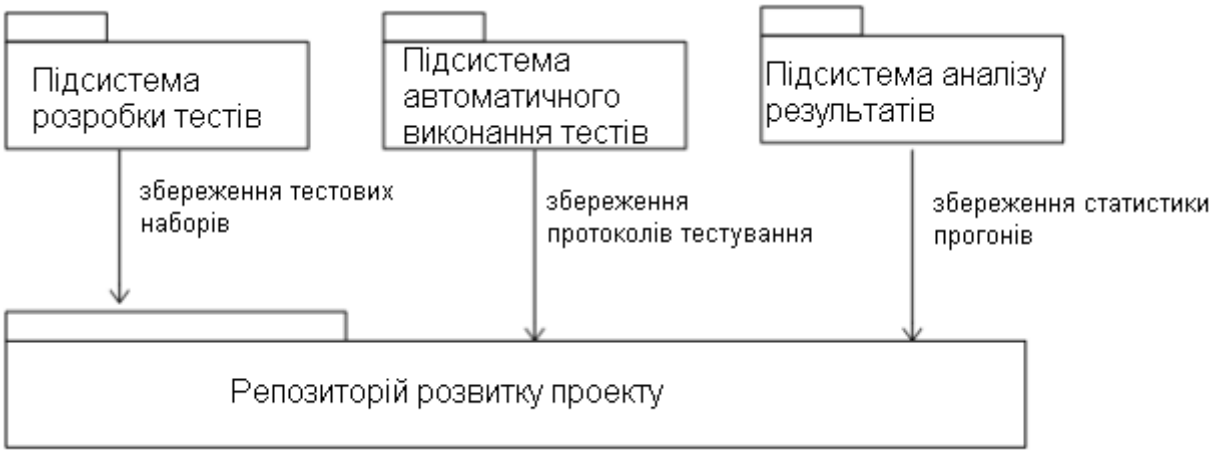


Рисунок 3.11 – Узагальнена структура системи автоматизації тестування.

Виправлена збірка программи надходить на наступний цикл тестування, і цикл повторюється, поки потрібну якість програмного комплексу не буде досягнуто. У цьому ітераційному процесі засоби автоматизації тестування забезпечують швидкий контроль результатів виправлення помилок і перевірку рівня якості, досягнутого в продукті [17].

Приклади позитивних і негативних тестів для аналізу коректності реєстрації користувача наведені нижче [19].

Приклади позитивних тестів:

1)

```
// С формы регистрации
// На почтовый ящик
// Логин минимальной длины (4 символа)
@Test(retryAnalyzer = Retry.class)
public void validRegformEmailMinloginDollarDayAddlimit() throws Exception
{

    cookieManager.setCookieLimit(true);
    app.getRegHelper().FillFormReg(
            false,
            new User().setLogin(GetRandomFull(4))
                    .setPassword(GetRandomFull(11)).setAgreement(true)
                    .setCheckEmail(true).setEmail(GetEmail()));
    Assert.assertTrue(app.getBaseMethod().isElementPresent(
```

2)

```
// С формы регистрации
// На почтовый ящик
// Пароль минимальной длины (8 символов)
@Test(retryAnalyzer = Retry.class)
public void validRegformEmailMinpassRublNolimit() throws Exception {
    cookieManager.setCookieLimit(false);
    app.getRegHelper().FillFormReg(
            false,
            new User().setLogin(GetRandomTextLogin() + "3")
                    .setPassword(GetRandomFull(8)).setAgreement(true)
                    .setCheckEmail(true).setEmail(GetEmail()));
    Assert.assertTrue(app.getBaseMethod().isElementPresent(

app.getRegHelper().pages.afterLogin.goToProfileFromPrivateOfficeH));
}
```

Приклади негативних тестів:

1)

```
// С формы регистрации
// На почтовый ящик
// Незаполнен логин
// Остальные параметры дефолтно
@Test(retryAnalyzer = Retry.class)
public void unvalidRegformEmailLoginisnull() throws Exception {
    app.getRegHelper().FillFormReg(
            new User().setPassword(GetRandomFull(11)).setAgreement(true)

                    .setCheckEmail(true).setEmail(GetEmail()));
    assertThat(
app.getRegHelper().pages.registerPage.getTextLoginFieldError(),
            equalToIgnoringCase("Имя пользователя должно содержать более 4
символов!"));
}
```

2)

```
// С формы регистрации
// На почтовый ящик
// Незаполнен пароль
// Остальные параметры дефолтно
@Test(retryAnalyzer = Retry.class)
public void unvalidRegformEmailPasswordisnull() throws Exception {
    app.getRegHelper().FillFormReg(
            new User().setLogin(GetRandomTextLogin()).setAgreement(true)
                    .setCheckEmail(true).setEmail(GetEmail()));
    assertThat(
app.getRegHelper().pages.registerPage.getTextPassFieldError(),
            equalToIgnoringCase("Введено менее 8 символов!"));
}
```

Нижче показані готові тести для тестування форми реєстрації.

```
// С формы регистрации
// На почтовый ящик
// Логин минимальной длины (4 символа)
// Долларовый счет
@Test
public void validRegformEmailMinloginDollarDayAddlimit() throws
Exception {
        app.getRegHelper().FillFormReg(
                false,
                new User().setLogin(GetRandomFull(4))

.setPassword(GetRandomFull(11)).setAgreement(true)

.setCheckEmail(true).setEmail(GetEmail()));
     Assert.assertTrue(app.getBaseMethod().IsElementPresent(

app.getRegHelper().pages.afterLogin.goToProfileFromAvatarB));
}

/    // С формы регистрации
// На почтовый ящик
// Пароль минимальной длины (8 символов)
// Без функционала выбора лимита
@Test
public void validRegformEmailMinpassRublNolimit() throws Exception
{
        app.getRegHelper().FillFormReg(
                false,
                new User().setLogin(GetRandomTextLogin())

.setPassword(GetRandomFull(8)).setAgreement(true)

.setCheckEmail(true).setEmail(GetEmail()));
     Assert.assertTrue(app.getBaseMethod().IsElementPresent(
```

```
      // С формы регистрации
      // Номер телефона +37
      @Test
      public void validRegformBelphoneRublMonthAddlimit() throws
Exception {
      cookieManager.setCookieLimit(true);
            app.getRegHelper().FillFormReg(
                        false,
                        new User().setLogin(GetRandomTextLogin())

      .setPassword(GetRandomFull(11)).setAgreement(true)
                                    .setCheckEmail(false)
                                    .setNumberPhone(GetNumberPhone(true))
                                    .setPhoneCode("+37"));
            Assert.assertTrue(app.getBaseMethod().IsElementPresent(

      app.getRegHelper().pages.afterLogin.goToProfileFromAvatarB));
      }
       // С формы регистрации
       // На почтовый ящик
       // Незаполнен логин
       // Остальные параметры дефолтно
       @Test
       public void unvalidRegformEmailLoginisnull() throws Exception {
             app.getRegHelper().FillFormReg(
                        new
 User().setPassword(GetRandomFull(11)).setAgreement(true)

       .setCheckEmail(true).setEmail(GetEmail()));
             assertThat(

       app.getRegHelper().pages.registerPage.getTextLoginFieldError(),
                              equalToIgnoringCase("Имя пользователя должно
       содержать более 4 символов!"));
       }
```

Порівняння автоматизованих і ручних тестів проводилося в середовищі Team City. Проект запуску тестів наведено на рисунку 3.12.

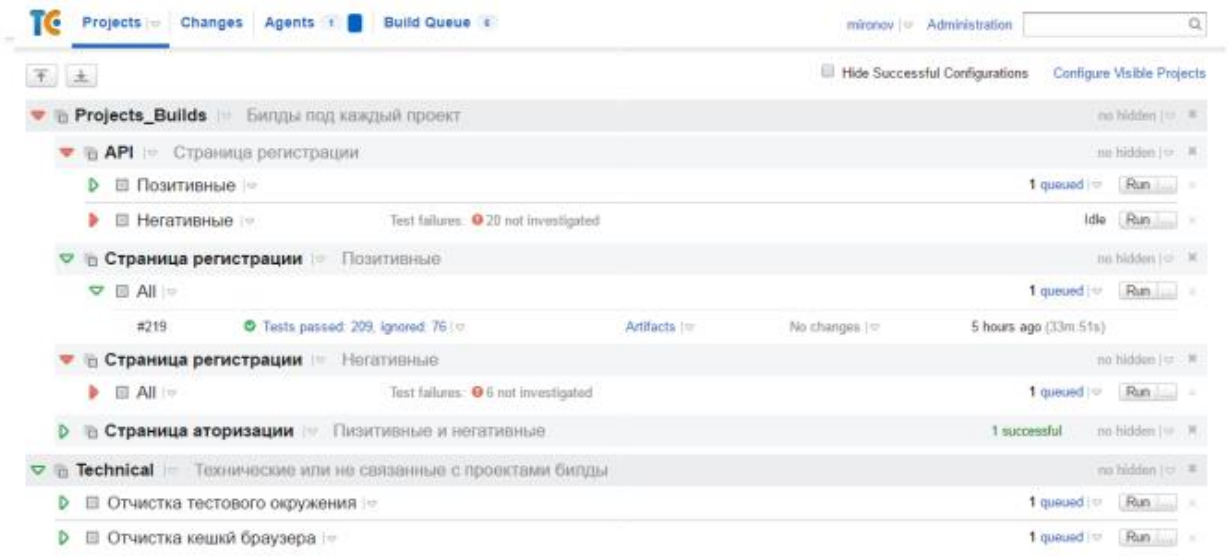


Рисунок 3.12 – Проект запуску тестів.

На рис. 3.12 зображено:

– API - тестування реєстрації через API-запит;

– три тестованих сайти;

– Technical - очищення тестового оточення перед прогоном тестів.

Дані для дослідження тимчасових характеристик ручного і автоматизованого тестування:

Комп'ютер: Процесор: Intel core i5-4430 CPU @ 3.00 GHz, ОЗУ: 8 гб;

Кількість тестів: до 500;

Середовище розробки тестів: Eclipse, Intellij IDEA.

Eclipse: 100 тестів пройшли за 35хв 58с.

Intellij IDEA: 100 тестів пройшли за 21 хв 12с.

Реальний час, якби тести виконувала людина: близько 24 годин (3 робочих дні).

Узагальнені результати аналізу часу тестування наведені на рисунку 3.13

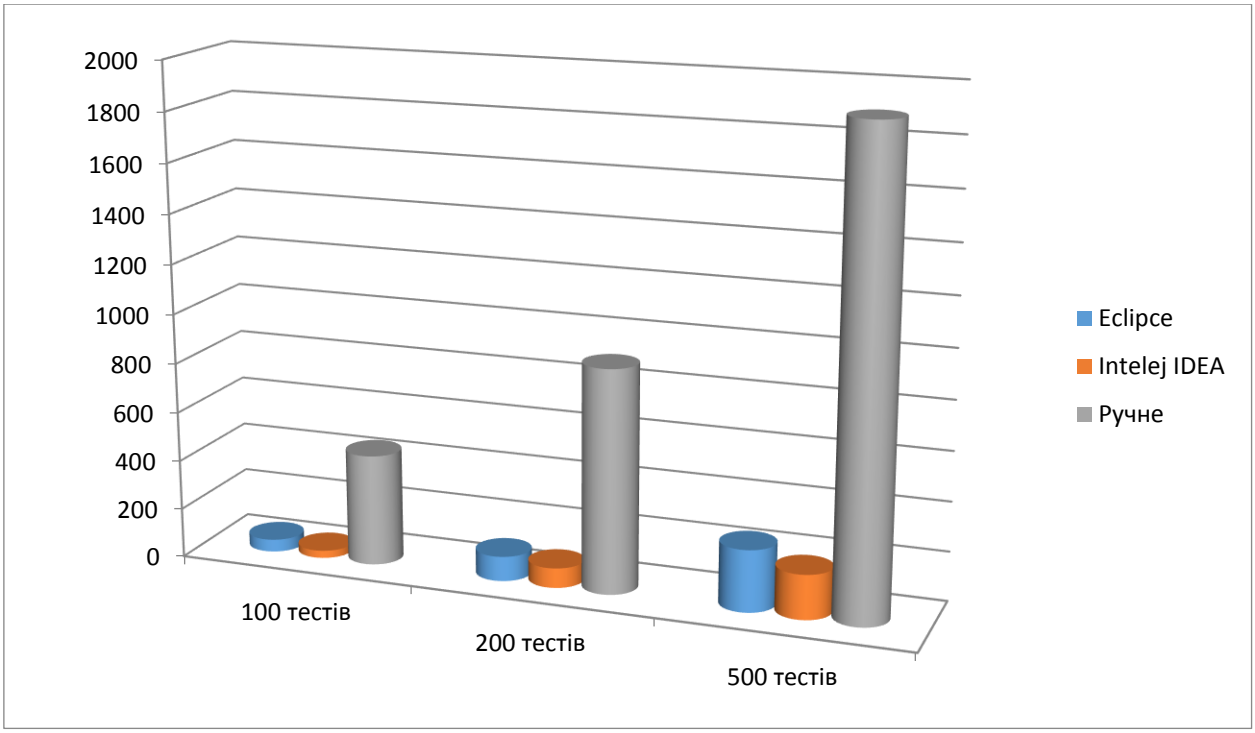


Рисунок 3.13 – Проект запуску тестів.

З рисунку 3.13 видно, що автоматизоване тестування в середовищі Intellij idea вимагає найменшого часу для виконання тестів.

Висновки до розділу

У рамках третього розділу здійснено проектування системи автоматизованого тестування веб-додатків, розроблено приклад тестування, перевірено продуктивність. На основі проведеного дослідження виявлено, що автоматизоване тестування в середовищі Intellij IDEA вимагає найменшого часу для виконання тестів.

ВИСНОВКИ

В ході виконання даної дипломної роботи було досліджено низку методів та алгоритмів автоматизації тестування веб-додатків.

Розкрито теорію тестування веб-додатків та проведено огляд засобів і методів автоматизації; наведено методології автоматичного тестування, а також запропоновано заходи, щодо підвищення продуктивності тестування.

Тестування проводиться з метою забезпечити якість розроблюваного веб-додатків. Основним параметром якості програми є надійність. Надійність визначається як ймовірність її роботи без відмов протягом певного періоду часу, розрахована з урахуванням вартості для користувача кожної відмови. Відмова веб-додатку – це прояв помилки в ньому. Звідси тестування веб-додатку – це процес виконання програми з метою виявлення в ньому помилок. "Вдалим" тестом є такий тест, на якому виконання програми завершилося з помилкою. Навпаки, "невдалим" називається тест, що не дозволив виявити помилку в програмі.

До основних плюсів автоматизації можна віднести можливість запускати скрипти в будь-який час доби на одній або декількох віддалених машинах, що дозволяє проводити автоматичне тестування паралельно з ручним.

З мінусів варто відзначити той факт, що на розробку і підтримку скриптів може йти досить багато часу, а також те, що далеко не вся функціональність піддається автоматизації.

На основі проведеного дослідження виявлено, що автоматизоване тестування в середовищі Intellij idea вимагає найменшого часу для виконання тестів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 1. Майерс Г. Мистецтво тестування програм: пер. з англ. / Майерс Г., Баджет Т., Сандлер К. – М.: Діалектика, 2012. – 270 с.

2. Кріспін Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд : пер. з англ. / Л. Кріспін., Д. Грегорі – М.: «Вільямс», 2011. – 463 с. – ISBN 0-471-46912-2.

3. Липаев В. В. Тестирование компонентов и комплексов программ: підруч. / Липаев В. В. – М.-Берлін: Діректт-Медіа, 2015. – 528с. – ISBN 978-5-89638-115- 0.

4. Азарский К. И. Тестирование. Легкий старт / Азарский К. И. – Mumai: Copyright, 2014. – 226 с. – ISBN 978-5-496-00893-8.

5. Лайза Криспин, Джанет Грегори. Гибкое тестирование. Вильямс, 2010. – 251 с.

6. Технология программирования. Основы современного тестирования программного обеспечения, разработанного на C#: учеб. пособие / под общ. ред. В.П. Котлярова. – СПб: СПбГПУ, 2004. – 168 с.

7. Каскадная модель // Википедия, свободная энциклопедия [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/Каскадная_модель/. – Дата доступа : 02.05.2019.

8. A brief Introduction to Scrum Methodology // MTC EduHub [Электронный ресурс]. – Режим доступа : https://mtceduhub.com/a-brief-introduction-to-scrum-methodology/. – Дата доступа : 02.05.2019.

9. Scrum // Википедия, свободная энциклопедия [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/Scrum/. – Дата доступа : 02.05.2019.

10. Мамонов, Д. Блеск и нищета автоматизации тестирования [Электронный ресурс] / Д. Мамонов. – СПб. : Wrike, 2017. – Режим доступа : https://habr.com/company/wrike/blog/321290/. – Дата доступа : 02.05.2019.

11. Черняк, М. Оценка эффективности автоматизации тестирования [Электронный ресурс] / М. Черняк. – 2015. – Режим доступа : http://www.a1qa.ru/blog/otsenka-effektivnosti-avtomatizatsiitestirovaniya/. – Дата доступа : 02.05.2019.

12. Шульга, В. Автоматизированное тестирование – «убийца» ручного тестирования, модный тренд или серебряная пуля? [Электронный ресурс] / В. Шульга. – 2018. – Режим доступа : https://habr.com/company/epam_systems/blog/349270/. – Дата доступа : 02.05.2019.

13. Вишневская Т.И., Романова Т.Н. Технология программирования: методические указа- ния к лабораторному практикуму. Ч. 1. – М: МГТУ им. Н.Э. Баумана, 2007. –58 с.

14. Макгрегор Дж., Сайкс Д. Тестирование объектно-ориентированного программного обеспечения: практическое пособие разработчикам, менеджерам проектов, программистам. – М: DiaSoft, 2002.– 432 с.

15. Вишневская Т.И., Романова Т.Н. Технология программирования: мет. указания к лабо- раторному практикуму. Ч. 2. – М: МГТУ им. Н.Э. Баумана, 2010. – 49 с.

16. C. Poole, J. Terrell, S. Busoli. NUnit 2012. [Электронный ресурс]. URL: http://www.nunit.org

17. XSpider 7.8 // PositiveTechnologies.2002-2012. [Электронный ресурс]. URL: http://www.ptsecurity.ru/xs7

18. MS Visual Studio Test Professional 2012 // Microsoft 2013. [Электронный ресурс]. URL: http://www.microsoft.com/visualstudio/rus/

19. Автоматизированное тестирование программного обеспечения, Элфрид Дастин, Джефф Рэшка, Джон Пол, Издательство: Лори, 2003 г.

20. Введение в тестирование программного обеспечения, Луиза Тамре, Вильямс, 2003

21. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем, Борис Бейзер, Питер, 2004 г.

22. Автоматизация процессов тестирования, И. Винниченко, Питер, 2005 г.

23. Тематические Медиа / Способ доступа: URL: http://habrahabr.ru/post/152653/. – "Что такое Selenium?".

24. The World Wide Web Consortium (W3C) / Спосіб доступу: URL: http://www.w3.org/TR/2012/WD-webdriver- 20120710/. – W3C Working Draft.

25. Software-Testing.Ru / Способ доступа: URL: http://software-testing.ru/library/testing/functional-testing/1740- what-is-webdriver. – " Что такое Selenium WebDriver?".

26. Bugs Catcher / Способ доступа: URL: http://bugscatcher.net/archives/1232. – WebDriver: Основные команды.

27. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. // СПб: Питер, 2004. – 344 с.

28. Винниченко И. Автоматизация процессов тестирования.// СПб, «Питер», 2005 – 437 с.

29. Дастин Э., Рэшка Д., Пол Д. \ Е. Молодцова, М. Павлов. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. // Москва, Издательство «ЛОРИ», 2003 – 435 с.

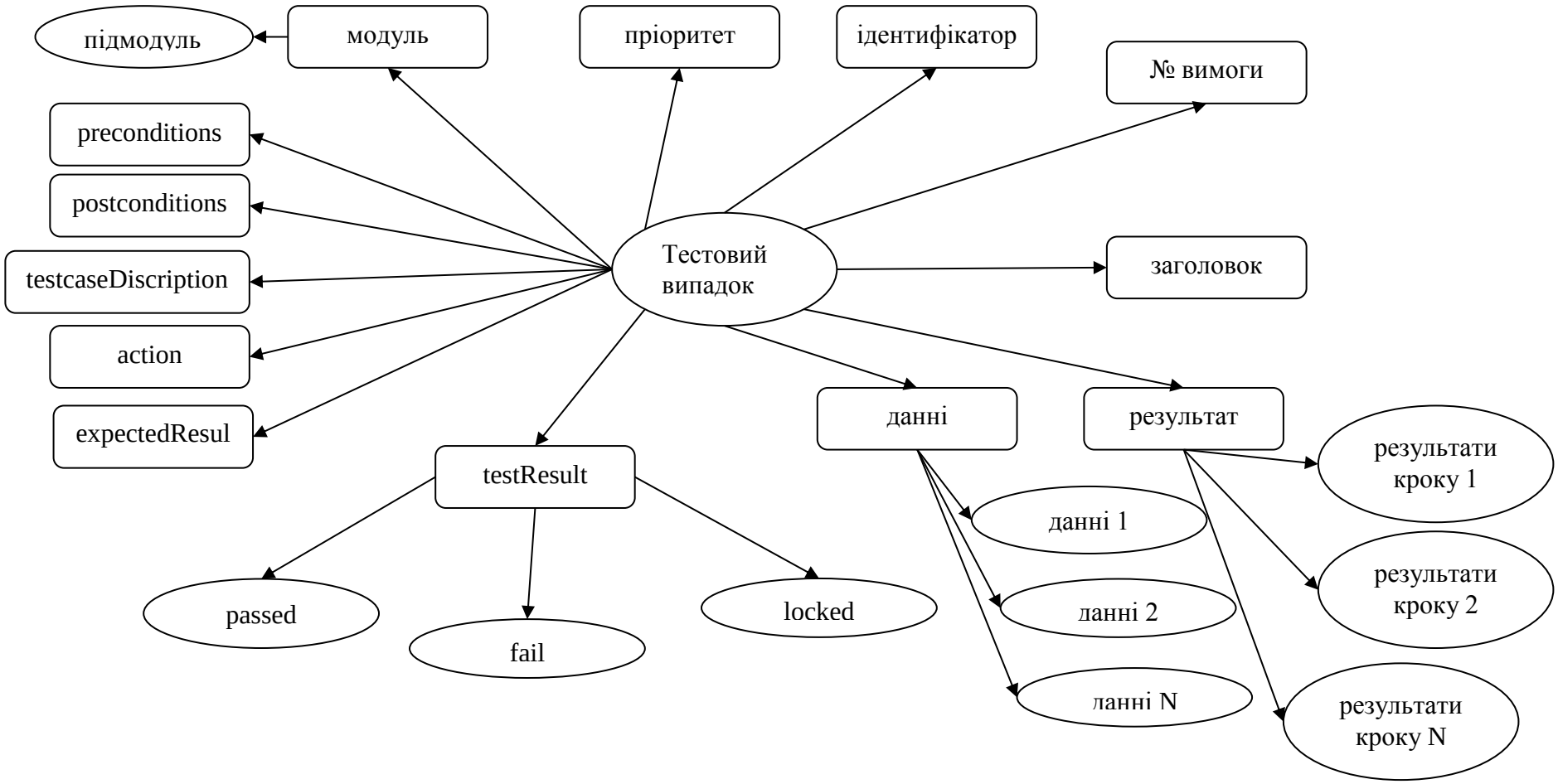
30. Котляров В.П., Коликова Т.В. Основы тестирования программного обеспечения. // СПб, Бином. Лаборатория знаний, 2006. – 321 с.

31. Полевой В. Как автоматизировать тестирование ПО? // Cnews, 2007. – 235 с.

32. Mark Fewster, Dorothy Graham. Software Test Automation. Effective use of test execution tools. // New York, ACM Press, 1999. – 123 p.

33. Алпаев Г. Учебник по Test Complete. http://tctutorial.ru/

ДОДАТОК А


підмодуль
модуль
пріоритет
ідентифікатор
№ вимоги
preconditions
postconditions
testcaseDiscription
action
expectedResul
Тестовий випадок
заголовок
данні
результат
testResult
результати кроку 1
данні 1
результати кроку 2
данні 2
passed
fail
locked
данні N
результати кроку N

ДОДАТОК Б

Приклад скрипта «allpairs.py»:

```
# Using AllPairs by MetaCommunications Engineering example.

from ext.metacomm.combinatorics import all_pairs2 as ap

def Generator(parameters):
    """
    Use generator which work on AllPairs-algorithm.
    Input:
    parameters - list of list of different parameters, that looks like this:
    [ [par1_val_1, par1_val_2, ...],
      [par2_val_1, par2_val_2, ...], ... ]
    Output:
    list of enumerate possible unique combinations, that looks like this:
    [ (0, [par1_val_1, ..., parK_val_K]), ...
      (N, [parX_val_X, ..., parY_val_Y]), ...]
    """
    combinations = list(enumerate(ap.all_pairs2(parameters)))
    return combinations


if __name__ == '__main__':
    # define list of list of different parameters:
    inputData = [
        ['Windows XP SP 3', 'Windows 7 SP 1', 'Debian 7.1', 'Ubuntu 12.04'],
        ['x86', 'x64'],
        ['chrome', 'firefox', 'safari']
        ]
    outputData = Generator(inputData)
    with open('output.txt', 'w') as fH:
        for line in outputData:
            stringData = 'Combination {}:\t{}'.format(str(line[0]),
str(line[1]))
            print(stringData)
            fH.write(stringData + '\n')
```